React Native

Composing Animations

Overview

- Goals of animation
  - Stationary objects must overcome inertia as they start moving.
  - Objects in motion have momentum and rarely come to a stop immediately.
  - Animations allow you to convey physically believable motion in your interface.

- React Native provides two complementary animation systems:
  - Animated for granular and interactive control of specific values
  - LayoutAnimation for animated global layout transactions.

Composing animations

- Animations can be combined and played in sequence or in parallel.
  - Sequential animations can play immediately after the previous animation has finished,
  - or they can start after a specified delay.
- The Animated API provides several methods which take an array of animations to execute and automatically call start()/stop() as needed.
  - sequence()
  - delay()
  - Etc.
- If one animation is stopped or interrupted, then all other animations in the group are also stopped.
  - **`Animated.parallel`** has a **`stopTogether`** option that can be set to false to disable this.

Composing reference: https://facebook.github.io/react-native/docs/animated#composing-animations

Composition functions

- Animated.delay(time) starts an animation after a given delay.
- Animated.parallel([array of animations]) starts a number of animations at the same time.
- Animated.sequence([array of animations]) starts the animations in order, waiting for each to complete before starting the next.
- Animated.stagger(time, [array of animations]) starts animations in order and in parallel, but with successive delays (delay time is determined by the time argument).

Composing reference: https://facebook.github.io/react-native/docs/animated#composing-animations

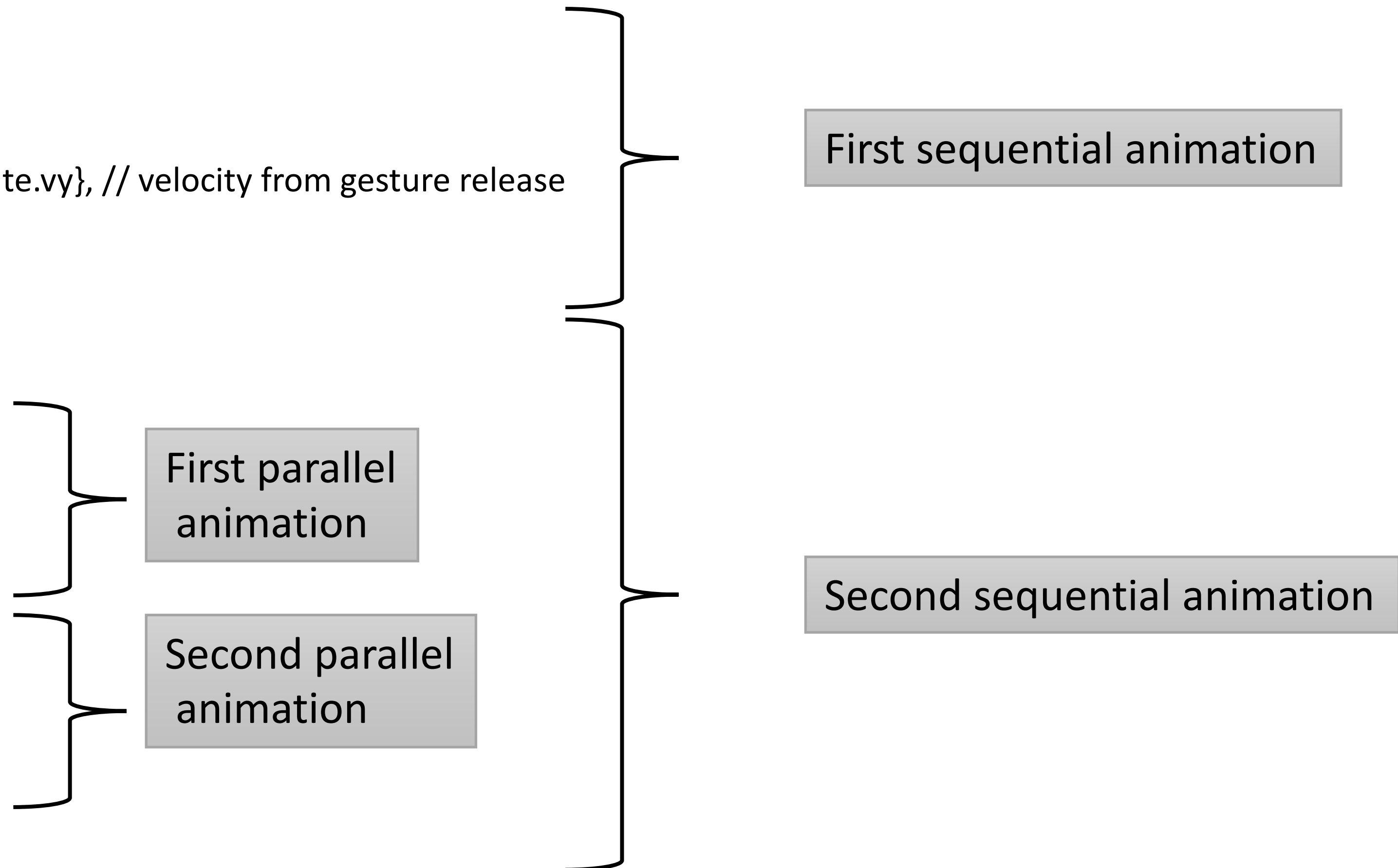
Composing

- Animations can also be chained together simply by setting the toValue of one animation to be another Animated.Value.
  - See Tracking dynamic values in the Animations guide.
- By default, if one animation is stopped or interrupted, then all other animations in the group are also stopped.

Composing animations

- example, the following animation coasts to a stop, then it springs back while twirling in parallel:

```
Animated.sequence([
  // decay, then spring to start and twirl
  Animated.decay(position, {
    // coast to a stop
    velocity: {x: gestureState.vx, y: gestureState.vy}, // velocity from gesture release
    deceleration: 0.997,
  }),
  Animated.parallel([
    // after decay, in parallel:
    Animated.spring(position, {
      toValue: {x: 0, y: 0}, // return to start
    }),
    Animated.timing(twirl, {
      // and twirl
      toValue: 360,
    }),
  ]),
]).start(); // start the sequence group
```



Example 1: transform and spin in sequence

This is anim5.js on the class website

```
import React from 'react';
import { Button, Animated, Text, View, Easing, Image, Dimensions } from 'react-native';

class MoveView extends React.Component {
  constructor(props){
    super(props);
    this.state = {
      animatedValue: new Animated.Value(0),  // Initial value for opacity: 0
      spinVal: new Animated.Value(0),  // Initial value for opacity: 0
      startX: 100,
      startY: 100,
      endX: Dimensions.get('window').width-100,
      endY: Dimensions.get('window').height-100,
    }
  }
```

State for animation. Note that there are two Animated.Values

Example 1: transform and spin in sequence

Animated.sequence will run when this function is called

```
doAnimate = () => {
  this.state.animatedValue.setValue(0);
  this.state.spinVal.setValue(0);
  Animated.sequence([
   Animated.timing(              // Animate over time
     this.state.animatedValue,          // The animated value to drive
    {
          toValue: 1,              // Animate to position: 1
          easing: Easing.back(2),
          duration: 9000,            // Make it take a while
    }
   ),
  Animated.timing(              // Animate over time
   this.state.spinVal,          // The animated value to drive
   {
          toValue: 1,              // Animate to position: 1
           easing: Easing.linear,
          duration: 9000,            // Make it take a while
   }
  )]
 ).start();                     // Starts the animation
}
```

Animated.sequence takes one parameter which is an array (angled bracket)

First animation. Note that it uses the first Animated.Value: animatedValue

Note the comma after the ending parenthesis of Animated.timing

Second animation. Note that it uses the second Animated.Value: spinVal

Note the bracket to end the array that is passed to Animated.sequence

We put the .start() after the closing parenthesis of the Animated.sequence function

Example 1: transform and spin in sequence

```
return (
 <View style={{flex:1}}>
 <View style={{flex:3}}>
  <Animated.Image              // Special animatable View
   style={{
    width: 227,
    height: 200,
    transform: [ {
       translateX: this.state.animatedValue.interpolate({
          inputRange: [0, 1],
          outputRange: [this.state.startX, this.state.endX]
     })},
     {
       translateY: this.state.animatedValue.interpolate({
          inputRange: [0, 1],
          outputRange: [this.state.startY, this.state.endY]
     })},] }}
    source={{uri:"https://s3.amazonaws.com/media-p.slides/uploads/alexanderfarennikov/images/1198519/reactjs.png"}}>
  </Animated.Image>
   </View>
```

Outer view

View around the first Animated.Image

Transform property: translateX. Note that it uses the first Animated.Value: animatedValue

Transform property: translateY. Note that it uses the first Animated.Value: animatedValue

End of the View around the **first** Animated.Image

Example 1: transform and spin in sequence

```
<View style={{flex:3, alignItems: 'center', justifyContent: 'center'}}>
 <Animated.Image            // Special animatable View
  style={{
   width: 227,
   height: 200,
   transform: [{rotate: spin}] }}
   source={{uri:"https://s3.amazonaws.com/media-p.slid.es/uploads/alexanderfarennikov/images/1198519/reactjs.png"}}>
 </Animated.Image>
 </View>
<View style={{flex:1}}>
 <Button
     onPress = {this.doAnimate}
     color="#841584"
     title="Click to animate"
     accessibilityLabel="Animation button"/>
  </View>

   </View>
); } }
```

View around the second Animated.Image

Transform property: **`rotate`**. Note that it uses the second Animated.Value: **`spinVal`**

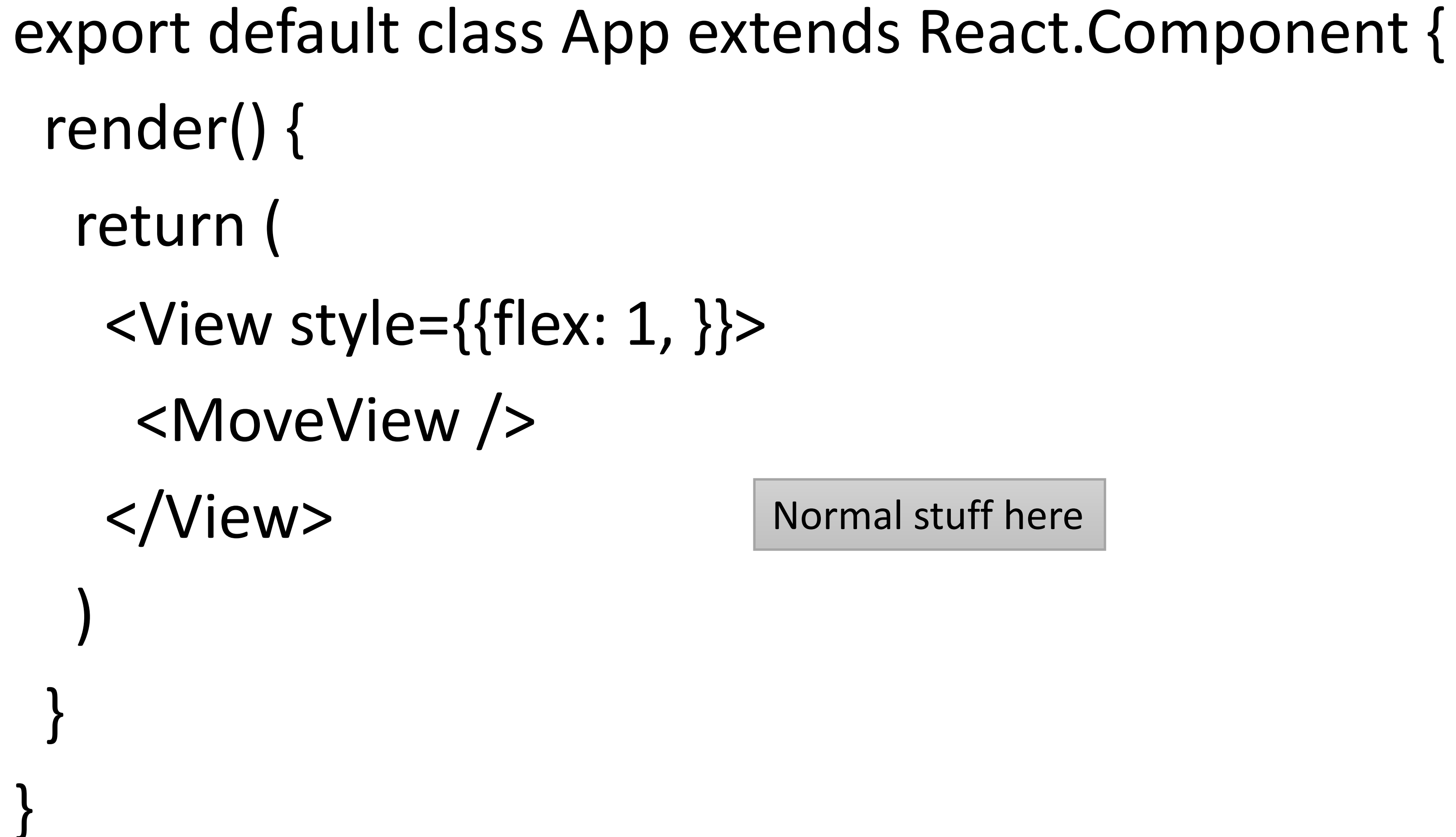
Button to start the animation sequence

End of the View around the second Animated.Image

End Outer view

Example 1: transform and spin in sequence

```
export default class App extends React.Component {
  render() {
    return (
      <View style={{flex: 1, }}>
        <MoveView />
      </View>
    )
  }
}
```



Example 2: two motion transforms staggered

- See example anim6.js on the class website.

Combining animated values

- There are some cases where an animated value needs to invert another animated value for calculation. An example is inverting a scale (2x --> 0.5x):
- You can combine two animated values via addition, subtraction, multiplication, division, or modulo to make a new animated value:
  - Animated.add()
  - Animated.subtract()
  - Animated.divide()
  - Animated.modulo()
  - Animated.multiply()

Combining animated values

- Example

```
const a = new Animated.Value(1);
const b = Animated.divide(1, a);

Animated.spring(a, {
  toValue: 2,
}).start();
```

Creating complex animations

```
import React from 'react';
import { Button, Animated, Text, View, Easing, Image, Dimensions } from 'react-native';

class MoveView extends React.Component {
  constructor(props){
    super(props);
    this.state = {
      animatedValue: new Animated.Value(0),  // Initial value for opacity: 0
      spinVal: new Animated.Value(0),  // Initial value for opacity: 0
      startX: 100,
      startY: 100,
      endX: Dimensions.get('window').width-100,
      endY: Dimensions.get('window').height-100,
    }
  }
```

Note that there are 2 different Animated Values

```
doAnimate = () => {
this.state.animatedValue.setValue(0);
this.state.spinVal.setValue(0);
  Animated.timing(                // Animate over time
   this.state.animatedValue,          // The animated value to drive
   {
    toValue: 1,               // Animate to position: 1
    easing: Easing.back(2),
    duration: 9000,            // Make it take a while
   }
  ).start();
  Animated.timing(                // Animate over time
   this.state.spinVal,           // The animated value to drive
   {
    toValue: 1,               // Animate to position: 1
    easing: Easing.linear,
    duration: 9000,            // Make it take a while
   }
 ).start();                  // Starts the animation
}
```

Reset the Animated Values so that the animation restarts

First animation. Note that it uses the first Animated Value And that it is started.

Second animation. Note that it uses the second Animated Value and that it is started.

Both animations are started in the doAnimate function.
Each animation works on a different Animated Value
But each Animated Value will be used by the same component (see next slide)

```
render() {

  const spin = this.state.spinVal.interpolate({
    inputRange: [0, 1],
    outputRange: ['0deg', '360deg']
  })
```

```
render() {
  const spin = this.state.spinVal.interpolate({
    inputRange: [0, 1],
    outputRange: ['0deg', '360deg']
  })
  return (
   <View style={{flex:1}}>
   <View style={{flex:3}}>
   <Animated.Image              // Special animatable View
    style={{
      width: 227,
      height: 200,
      transform: [ {
          translateX: this.state.animatedValue.interpolate({
              inputRange: [0, 1],
              outputRange: [this.state.startX, this.state.endX]
       })},
```

This Animated.Image will be animated.

The first transformation will translate both X and Y. This is continued on the next slide

```
{
   translateY: this.state.animatedValue.interpolate({
     inputRange: [0, 1],
     outputRange: [this.state.startY, this.state.endY]
}))},
{rotate:spin}
 ] }}
  source={{uri:"https://s3.amazonaws.com/media-p.slid.es/uploads/alexanderfarennikov/images/1198519/reactjs.png"}}>
 </Animated.Image>
  </View>
 <View style={{flex:1}}>
  <Button
     onPress = {this.doAnimate}
     color="#841584"
     title="Click to animate"
     accessibilityLabel="Animation button"/>
  </View>
</View>);
} }
```

End of the first transformation using the first Animated Value

This is the spin transformation using the second Animated Value

The transform property of the style takes an array. The array in this program has three elements, the translateX, translate, and rotate elements.

```
export default class App extends React.Component {
  render() {
    return (
      <View style={{flex: 1, }}>
        <MoveView />
      </View>
    )
  }
}
```

LayoutAnimation API

- `LayoutAnimation` allows you to globally configure create and update animations that will be used for all views in the next render/layout cycle.
  - useful for doing flexbox layout updates without bothering to measure or calculate specific properties in order to animate them directly,
  - especially useful when layout changes may affect ancestors,
  - example: a "see more" expansion that also increases the size of the parent and pushes down the row below which would otherwise require explicit coordination between the components in order to animate them all in sync.

- provides much less control than Animated and other animation libraries,
  - may need to use another approach if you can't get `LayoutAnimation` to do what you want.

LayoutAnimation API

- in order to get this to work on **Android** you need to set the following flags via UIManager:

```
UIManager.setLayoutAnimationEnabledExperimental &&
      UIManager.setLayoutAnimationEnabledExperimental(true);
```

Example 2: LayoutAnimation

```
import React from 'react';
import {
  NativeModules,
  LayoutAnimation,
  Text,
  TouchableOpacity,
  StyleSheet,
  View,
} from 'react-native';

const { UIManager } = NativeModules;

UIManager.setLayoutAnimationEnabledExperimental &&
  UIManager.setLayoutAnimationEnabledExperimental(true);
```

anim8.js on class website

```
export default class App extends React.Component {
  state = {
    w: 100,
    h: 100,
  };

  _onPress = () => {
    // Animate the update
    LayoutAnimation.spring();
    this.setState({w: this.state.w + 15, h: this.state.h + 15})
  }
```

Other animation options are **`easeInEaseOut`** and **`linear`**

Animation automatically run when layout is next rendered.

This will force a new render.

```
render() {
  return (
    <View style={styles.container}>
      <View style={[styles.box, {width: this.state.w, height: this.state.h}]} />
      <TouchableOpacity onPress={this._onPress}>
        <View style={styles.button}>
          <Text style={styles.buttonText}>Press me!</Text>
        </View>
      </TouchableOpacity>
    </View>
  );
 }
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  box: {
    width: 200,
    height: 200,
    backgroundColor: 'red',
  },
  button: {
    backgroundColor: 'black',
    paddingHorizontal: 20,
    paddingVertical: 15,
    marginTop: 15,
  },
  buttonText: {
    color: '#fff',
    fontWeight: 'bold',
  },
});
```