Contents

Preface

This book provides a way to learn `Wheel.js` by exploring it's contepts and providing examples of common `Wheel.js` patterns and implementation of common interface interaction patters with `Wheel.js` .

Chapter 1

Step by Step Concepts of Wheel.js

To illuminate the concepts of Wheel.js

1.1 What is Wheel.js?

1.2 What is an App?

1.3 Creating a View

1.4 What is dollar?

1.5 Simple Event Handling

1.6 Supported Events

Chapter 2

Examples of Wheel.js

To provide examples of Wheel.js

2.1 Smooth Drag Event Example

Implementing Drag Events in Wheel involves setting up listeners which will fire in 2 stages.

```
60          listen:function() {
61
62            this.$.on('tapstart' ,
63                      this.triggerDragInit.bind(this));
64
65            this.$.on('dragstart' ,
66                      this.onDragStart.bind(this));
67
68            this.$.on('dragmove' ,
69                      this.onDragMove.bind(this));
70
71            this.$.on('dragend' ,
72                      this.onDragFinished.bind(this));
73          },
```

The first stage is to know what kind of event you would like to initiate the drag.

In my example I have chosen *tapstart* (which is both 'mousedown' and 'touchstart') this gives the effect of 'click-and-hold' for drag on desktop and the effect of 'touch-and-hold' on touch devices.

The second stage is to know that wheel understands 4 events related to drag.

draginit

1. This is an event that is triggered in the callback event of and initiating event. The callback snippet below demonstrates using the *$* dom-property to invoke the *draginit* event.

```
75          triggerDragInit: function(e) {
76            // enable successive dragmove events on motion
77            this.$.trigger('draginit');
78          },
```

dragstart

1. Once *draginit* has been triggered - *dragstart* will be called by the Wheel framework.

```
80          onDragStart: function(e) {
81            this.touchOriginX = e.pageX;
82            this.touchOriginY = e.pageY;
83          },
```

dragmove

1. The Wheel framework will generate *dragmove* events on movement of the still-touching finger - or the still held down mouse. The position of the finger or mouse respectively is encoded in 'pageX' and 'pageY'.

```
85          onDragMove: function(e) {
86            this.assignNextBoxTravel(e.pageX, e.pageY);
87          },
```

dragend

1. This is triggered when Wheel detects the opposite of your initiate drag event. In this case it will be *touchstop*

```
89          onDragFinished: function(e) {
90            this.currentBoxCornerLeft = this.nextBoxCornerLeft;
91            this.currentBoxCornerTop = this.nextBoxCornerTop;
92          },
```

I am going to explain *dragstart*, *dragmove* and *dragend* in terms of STARTING, DYNAMIC, and FINAL states and how they can be used to implement a smooth drag algorithm. To ease explaining I will refer the the mobile interaction with the touching finger - on a mobile device the finger shares most properties of a desktop mouse being moved with a left-click held.

In the below diagram I have detailed these 3 'states' - STARTING, DYNAMIC, and FINAL – I will explain them following the diagram.

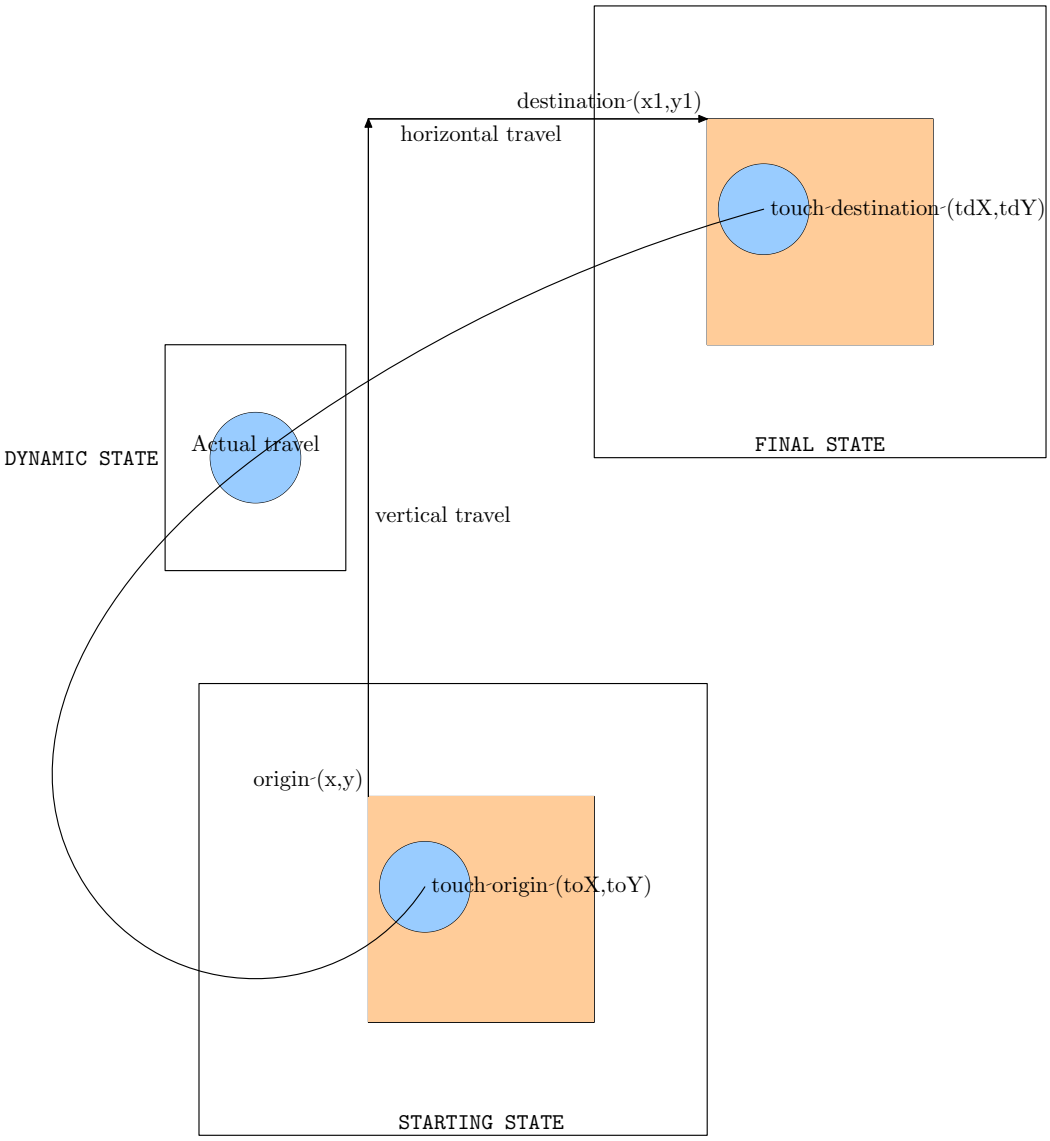

Figure 2.1: Implementing Smooth Drag in Wheel.js

STARTING state is used to record the initial touch origin of the finger press. These inital values are used as the basis for calculating 'horizontal travel' and 'vertical travel'. In wheel this state is realized by using *dragstart*

```
80          onDragStart: function(e) {
81            this.touchOriginX = e.pageX;
82            this.touchOriginY = e.pageY;
83          },
```

DYNAMIC state is when the finger is held on the screen and moved around. In wheel this state is realized by using *dragmove*

Wheel normalizes what would generally be clientX/clientY or pageX/pageY in various browsers into pageX/pageY.

```
85        onDragMove: function(e) {
86          this.assignNextBoxTravel(e.pageX, e.pageY);
87        },
```

Wheel also has the faculty to provide deltaX and deltaY properties as part of an event. Converting this code to use this simplifying faculty of Wheel is left as an excersize to the reader.

For the purposes of this example we are calculating deltaX and deltaY manually to expose the relationships that can be utilized by harnessing *dragstart*, *dragmove* and *dragend* together. I refer to deltaX as horizontalTravel and deltaY as verticalTravel in the example.

This function details the process of calculating horizontalTravel and verticalTravel from the value of pageX and pageY. The horizontal travel and the vertical travel are relative to the touch position recorded in *dragstart*.

```
94        assignNextBoxTravel:function(nextTouchDownX,nextTouchDownY) {
95          var horizontalTravelPx = nextTouchDownX - this.touchOriginX;
96          var verticalTravelPx   = nextTouchDownY - this.touchOriginY;
```

The current postion of the box that the finger is moving is based on the value that is stored in the View object. The initial values are 0 and 0 for x and y respectively. This current position becomes the next position by including the values of horizontal travel and vertical travel to x and y respectively.

```
98          this.nextBoxCornerLeft = this.currentBoxCornerLeft + horizontalTravelPx;
99          this.nextBoxCornerTop  = this.currentBoxCornerTop + verticalTravelPx;
```

The $ operator is used to alter the css style attribute with this next value for top and left. This creates the illusion that the box is moving with the finger.

```
101         this.$.css('left', this.nextBoxCornerLeft);
102         this.$.css('top' , this.nextBoxCornerTop);
103       }
```

FINAL state is when the finger is released. In wheel this state is realized by using *dragend*. The values from the previous state dynamic will be commited to the object as the current position of the box that the view represents. Because the dom has already been updated in the DYNAMIC state - what you see is what the object knows as-well.

```
89        onDragFinished: function(e) {
90          this.currentBoxCornerLeft = this.nextBoxCornerLeft;
91          this.currentBoxCornerTop = this.nextBoxCornerTop;
92        },
```

The complete code listings for this example is as follows :

```
<html>
  <head>
    <!-- give myself a consistant element style model to work with -->
    <link rel="stylesheet" href="reset.css">
    <!-- mobile friendly : single page app - no zooming - device sensitive -->
    <meta name="viewport"
          content="width=device-width,
                   initial-scale=1.0,
                   maximum-scale=1.0,
                   minimum-scale=1.0">

    <title> Smooth Drag Patch </title>
    <style>
      body {
        background-color:#A0A0A0;
        position:relative;
      }
      .drag-patch {
        position         : absolute;
        width            : 149px;
        height           : 149px;
        background-color : #8833A0;
        border-color     : #007700;
        border-width     : 1px;
        border-style     : solid;
      }
    </style>
  </head>

  <body>
    <div id="main">
    </div>

    <script type="text/javascript"
            src="wheel-modern-standalone-0.3.1.js">
    </script>

    <script class='template'
            type="text/html"
            name='App.Drag.Patch'>
      <div class='drag-patch'>
      </div>
    </script>

    <script type="text/javascript">
      Wheel.App.subclass('App', {
        init: function() {
          App.Drag.Patch.build({parent:$('#main')});
        }
      });

      Wheel.View.subclass('App.Drag.Patch', {
        init  :function() {
          this.currentBoxCornerTop  = 0;
          this.currentBoxCornerLeft = 0;
          this.$.css('left', this.currentBoxCornerLeft);
          this.$.css('top', this.currentBoxCornerTop);
        },

        listen:function() {

```

```
62            this.$.on('tapstart' ,
63                      this.triggerDragInit.bind(this));
64
65            this.$.on('dragstart' ,
66                      this.onDragStart.bind(this));
67
68            this.$.on('dragmove' ,
69                      this.onDragMove.bind(this));
70
71            this.$.on('dragend' ,
72                      this.onDragFinished.bind(this));
73          },
74
75          triggerDragInit: function(e) {
76            // enable successive dragmove events on motion
77            this.$.trigger('draginit');
78          },
79
80          onDragStart: function(e) {
81            this.touchOriginX = e.pageX;
82            this.touchOriginY = e.pageY;
83          },
84
85          onDragMove: function(e) {
86            this.assignNextBoxTravel(e.pageX, e.pageY);
87          },
88
89          onDragFinished: function(e) {
90            this.currentBoxCornerLeft = this.nextBoxCornerLeft;
91            this.currentBoxCornerTop = this.nextBoxCornerTop;
92          },
93
94          assignNextBoxTravel:function(nextTouchDownX,nextTouchDownY) {
95            var horizontalTravelPx = nextTouchDownX - this.touchOriginX;
96            var verticalTravelPx   = nextTouchDownY - this.touchOriginY;
97
98            this.nextBoxCornerLeft = this.currentBoxCornerLeft + horizontalTravelPx;
99            this.nextBoxCornerTop  = this.currentBoxCornerTop + verticalTravelPx;
100
101           this.$.css('left', this.nextBoxCornerLeft);
102           this.$.css('top' , this.nextBoxCornerTop);
103         }
104       });
105
106       App.build();
107
108     </script>
109   </body>
110 </html>
```

Bibliography

[1] L. Lamport. **LaTeX A Document Preparation System** Addison-Wesley, California 1986.

[2] CTJ. Dodson. **LaTeX PDF LaTeX by Example** http://www.maths.manchester.ac.uk/ kd/latextut/pdfbyex.htm, 2001.