Pointer Events

on May, 2019 I wrote a post about Pointer Events as an introduction to Pointer Events and how they could be polyfilled for browsers (Safari) that didn't support them natively.

Almost a year later things have progressed. Safari supports the feature so there's no need for a polyfill and it's now practical to use in production.

As a refresher, these are the events, the onEvent handlers and a brief explanation of when they trigger, taken from MDN

Event	On Event Handler	Description
`pointerover`	`onpointerover`	Fired when a pointer is moved into an element's hit test boundaries.
`pointerenter`	`onpointerenter`	Fired when a pointer is moved into the hit test boundaries of an element or one of its descendants, including as a result of a pointerdown event from a device that does not support hover (see `pointerdown`).
`pointerdown`	`onpointerdown`	Fired when a pointer becomes *active buttons state*.
`pointermove`	`onpointermove`	Fired when a pointer changes coordinates. This event is also used if the change in pointer state can not be reported by other events.
`pointerup`	`onpointerup`	Fired when a pointer is no longer *active buttons state*.
`pointercancel`	`onpointercancel`	A browser fires this event if it concludes the pointer will no longer be able to generate events (for example the related device is deactived).

Event	On Event Handler	Description
pointerout	onpointerout	Fired for several reasons including: pointer is moved out of the hit test boundaries of an element; firing the pointerup event for a device that does not support hover (see pointerup); after firing the `pointercancel` event (see `pointercancel`); when a pen stylus leaves the hover range detectable by the digitizer.
pointerleave	onpointerleave	Fired when a pointer is moved out of the hit test boundaries of an element. For pen devices, this event is fired when the stylus leaves the hover range detectable by the digitizer.
gotpointercapture	ongotpointercapture	Fired when an element receives pointer capture.
lostpointercapture	onlostpointercapture	Fired after pointer capture is released for a pointer.

The idea is to recreate two basic event handlers so they'll work across devices: click and hover.

First, naive implementation

Using the following HTML code.

```
<div class="box">
  <h1>Click Me!</h2>
</div>
```

We can use the following Javascript code to listen for `pointerdown` and `pointerover` using code like the one below.

```
if (window.PointerEvent) {
```

```
  const box = document.querySelector(".box");

  box.addEventListener("pointerdown", (evt) => {
    console.log("Pointer click equivalent");
  });

  box.addEventListener("pointerover", (evt) => {
    console.log("Pointer moved in");
  });
}
```

We wrap our code on a basic feature detection block to make sure we only use the feature in browsers that support it.

Next, we capture a reference to the HTML object that we want to work with and add the two event listeners.

Most of the time this will be OK as the behavior we want is similar across pointing devices

Take two

But there are times when we may want to do things differently based on what type of device is accessing the element and how it's doing it.

For example, it's different to click on a button with your finger than with a pen.

The `pointerover` event remains the same as that one doesn't need to know the type of pointing device that we used.

We change `pointerdown` to a switch statement where we test for different types of pointer devices and take appropriate action based on the type of device.

We use a switch statement to match the type of pointer in use: mouse, pen, or touch.

```
const box = document.querySelector(".box");
```

```
if (window.PointerEvent) {
  box.addEventListener("pointerover", (evt) => {
    console.log("Pointer moved in");
  });

  box.addEventListener("pointerdown", (evt) => {
    switch(evt.pointerType) {
      case "mouse":
        console.log('mouse input detected');
        break;
      case "pen":
        console.log('pen/stylus input detected');
        break;
      case "touch":
        console.log('touch input detected');
        break;
      default:
        console.log('pointerType is empty or could not be detected');
    }
  });
}
```

Refining the code

The last portion of this post will cover some refinements that we can do to the script to improve performance.

First we create external functions for each of the events that we want to handle.

```
function handlePointerOver(evt) {
  console.log("Pointer moved in");
}

function handlePenInput(evt) {
  console.log("pen/stylus input detected");
}
```

```
function handleTouchInput(evt) {
  console.log("touch input detected");
}

function handleMouseInput(evt) {
  console.log("mouse input detected");
}
```

Then we reference the functions from inside the event handlers. This way we make the code more modular.

```
if (window.PointerEvent) {
  box.addEventListener("pointerover", (evt) => {
    handlePointerEvent(evt);
  });

  box.addEventListener("pointerdown", (evt) => {
    console.log("Pointer is down");
    switch (evt.pointerType) {
      case "mouse":
        handleMouseInput(evt);
        break;
      case "pen":
        handlePenInput(evt);
        break;
      case "touch":
        handleTouchInput(evt);
        break;
      default:
        console.log("pointerType could not be detected");
    }
  });
}
```

We can further refine the touch event handler by using multi-touch interactions as explained in MDN's Multi-touch interaction