canvasInteractor

Make your HTML canvas Interactive

Stefan Gössner[1],

[1]Dortmund University of Applied Sciences. Department of Mechanical Engineering

April 2022

Keywords: canvasInteractor, canvas, javascript, events, requestAnimationFrame, throttling, event loop

1. What is It ?

`canvasInteractor` is a JavaScript micro-library (9.1 kB uncompressed) used to handle pointer events for simple geometry editing together with one or more HTML canvases [1]. It implements a global event loop based on `requestAnimationFrame` and supports throttling of `pointermove` and `wheel` events via its custom `tick` event for efficient animation [2]. Cartesian coordinates with user defined origin are possible.

It was primarily implemented for use in engineering education and conference presentations.

`canvasInteractor` is the modern and more minimal successor of deprecated `canvas-area` [3].

2. How to Initialize ?

For each HTML canvas in an HTML document an instance via `canvasInteractor.create` is required.

```
<canvas id="c" width="601" height="401"></canvas>
<script src="https://cdn.jsdelivr.net/gh/goessner/canvasinteractor/canvasInteractor.js">
</script>
<script>
    const ctx = document.getElementById('c').getContext('2d');
    const interactor = canvasInteractor.create(ctx, {x: 300,  // view ...
                                                     y: 200,  // ... properties
                                                     cartesian: true});

    // ...
</script>
```

Listing 1: Constructor – cartesian coordinates with origin centered.

View coordinates provided by events can be controlled in the constructor by an additional view argument `{x=0,y=0,scl=1,cartesian=false}` beside the canvas `RenderingContext2D` object.

- `x,y` … view's origin location.
- `scl` … view's scaling.
- `cartesian` … cartesian coordinate system (y-axis up).

3. Handling Events

Each `canvasInteractor` instance handles DOM pointer events and some custom events.

Table 1: Supported Events

Event	Comment
`pointermove`	Pointer moved.
`pointerdown`	Pointer device button pressed.
`pointerup`	Pointer device button released.
`pointerenter`	Pointer enters canvas.
`pointerleave`	Pointer leaves canvas.
`pointercancel`	DOM event forwarded.
`click`	Pointer device button pressed and released at the same location.
`wheel`	Pointer device wheel event.
`tick`	Throttled timer event. At most every 60 milliseconds.
`pan`	Pan by pointer device. Occuring only with (left) button pressed.
`drag`	Drag by pointer device. Occuring only with (left) button pressed and something was signalled as 'hit'.

Instead of registering events via well known `addEventListener`, an application registers events to a `canvasInteractor` instance using thats `on` method.

```
// ...
interactor.on('pointermove', (e) => { /* do stuff */ })
          .on('tick', (e) => { /* do other stuff */ })
          .on('pan',  (e) => { /* do yet other stuff */ })
          .startTimer();
```

Listing 2: Registering events and starting tick timer.

The `pointermove` event in combination with (left) pointer device button pressed also notifies the application of a `drag` or `pan` event, whether an underlying element is `hit` or not. See the example how to control that behavior.

Callback functions registered via `on` recieve an extended event object `e`.

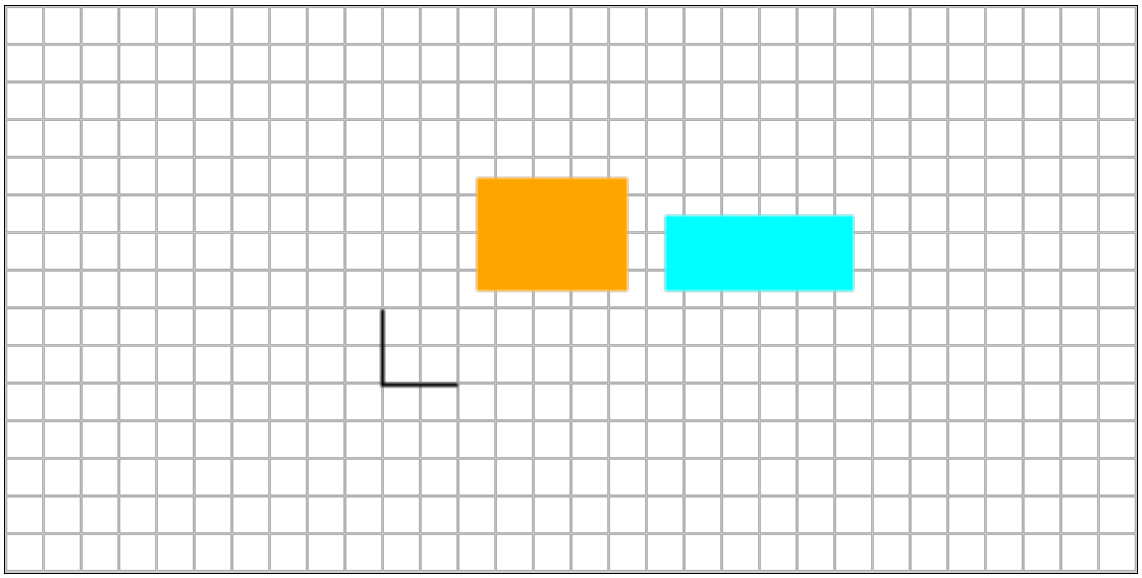
Table 2: Event object properties

Property	Type	Comment
`type`	`string`	Event type.
`x`, `y`	`number`	Canvas coordinates with respect to upper left or lower left (`cartesian`) corner.
`dx`, `dy`	`number`	Pointer location displacement from previous position.
`xusr`, `yusr`	`number`	Coordinates with respect to user defined view origin and scaling.
`dxusr`, `dyusr`	`number`	Pointer location displacement with respect to user defined view scaling.

Property	Type	Comment
`btn`	`number`	Pointer device button identifier on button press (left: `1`, right: `2`, middle: `4`).
`dbtn`	`number`	Pointer device button difference on button release (left: `-1`, right: `-2`, middle: `-4`).
`eps`	`number`	Some pixel tolerance for selecting/hitting (default = `5`).
`inside`	`boolean`	Is pointer currently inside canvas.
`delta`	`number`	Wheel delta.
`hit`	`boolean`	Needs to be set by application within `pointermove` event. Can be treated then within `tick` event.

4. Example

The example shows how to use `canvasInteractor`. Rectangles can be `drag`ged, whereas the origin symbol can not, which induces the `pan` event. `zoom`ing is done by the pointer device' wheel operation.



fps: 60 | ☑ cartesian | zoom-scale: 1.00 | pos: -2 / -2 | action: - |

```
<!doctype html>
<html>
<head>
    <title>canvasInteractor example</title>
</head>

<body>
    <h1>canvasInteractor example</h1>
    <canvas id="c" width="601" height="301"
            style="border:1px solid black;background-color:snow"></canvas>

    <script src="https://cdn.jsdelivr.net/gh/goessner/canvasinteractor/canvasInteractor.js">
</script>
    <script>
    const ctx = document.getElementById('c').getContext('2d');
    const interactor = canvasInteractor.create(ctx, {x:200,
                                                     y:100,
                                                     scl:1,
                                                     cartesian:true});
    const rec1 = {x:50,y:50,b:80,h:60,fs:'orange',lw:4};
    const rec2 = {x:150,y:50,b:100,h:40,fs:'cyan',lw:4};

    function render() {
        // ...
    }

    function hitRec(x,y,rec) {
        return (x > rec.x && x < rec.x + rec.b && y > rec.y && y < rec.y + rec.h);
    }
    interactor
        .on('tick', (e) => {
            render();
        })
        .on('pointermove', (e) => {
            rec1.sel = hitRec(e.xusr,e.yusr,rec1) ? true : false;
            rec2.sel = hitRec(e.xusr,e.yusr,rec2) ? true : false;
            e.hit = rec1.sel || rec2.sel;
        })
        .on('wheel',  (e) => {   // zooming about pointer location ...
            interactor.view.x = e.x + e.dscl*(interactor.view.x - e.x);
            interactor.view.y = e.y + e.dscl*(interactor.view.y - e.y);
            interactor.view.scl *= e.dscl;
        })
        .on('pan',  (e) => {
            interactor.view.x += e.dx;
            interactor.view.y += e.dy;
        })
        .on('drag', (e) => {
            if (rec1.sel) { rec1.x += e.dxusr; rec1.y += e.dyusr; }
            if (rec2.sel) { rec2.x += e.dxusr; rec2.y += e.dyusr; }
        })
        .startTimer();
    </script>
</body>
</html>
```

Listing 3: canvasInteractor example.

5. Conclusion

`canvasInteractor` is a tiny JavaScript library enhancing and extending HTML canvas' event handling for performant animation and geometrical interaction.

A global event loop (singleton) based on `requestAnimationFrame` provides assistance with event throttling via its custom `tick` event. Cartesian coordinates preferred in scientific and engineering applications are supported. *Pan*, *drag* and *zoom* based on user defined origins can be done.

References

[1] Canvas API, https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
[2] D. Corbacho, Debouncing and Throttling Explained Through Examples
https://css-tricks.com/debouncing-throttling-explained-examples/
[3] S. Goessner, `canvas-area`, https://github.com/goessner/canvas-area
[4] S. Goessner, `canvasInteractor`, https://github.com/goessner/canvasInteractor