ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP036

TOUCH-SCREEN WEB INTERFACES

L. Zambon*, A. Apollonio, R. Passuello, Elettra Sincrotrone Trieste, Trieste, Italy

Abstract

A touch screen (mobile or not mobile) has a significant impact on the kind of interaction between humans and control systems. This paper describes the development of some widgets and applications based on touch screens. The technologies used (for example PUMA, JavaScript and SVG) will be discussed in detail. Also a few tests and use-cases will be described compared with normal screens, mouse and keyboard interaction.

INTRODUCTION

A prototype of touch screens was patented in 1946 but have become familiar to users only since the diffusion of smartphones and tablets.

Our development is based mainly on 2 JavaScript libraries: Hammer.js [1] for 2D applications, Three.js [2] for 3D models.

Data are taken from the Control System thorough PUMA [3]

2D APPLICATIONS

Hammer.js is a JavaScript library which efficiently captures gestures. Gestures like tap pinch etc are captured very quickly and are assimilated to the events produced by mouse. Any gesture also produces the event start, continue and stop. All aspects can be effectively customized. By default all gestures have a correspondence in mouse events (for example mouse wheel or right click) so the same web interface can be used on both touch and not touch screens. Hammer.js was used in our project in conjunction with SVG (Scalable Vector Graphics) [4]. SVG has been developed by W3C since1999. It is an XML based 2D vectorial graphic format, it recognizes paradigms very similar to CSS and elements can be addressed by JavaScript like HTML elements.

SVG can be considered as "the" graphic extension of HTML. Graphic elements can be grouped in a symbol. A symbol can be repeated many times and can be customized (for example the filling color) as a unity. Every object can be translated by a transformation matrix or by a simple transformation command like translate or rotate. A significant trick in building circular design was to generate each element in the upper central position and then rotate it by a variable angle.

The Knob

The "Knob" (Fig. 1) is a component which allows setting a value with a user experience similar to a physical knob. It is composed of two wheels with different colors.

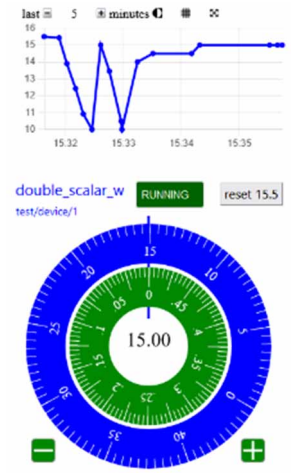

Figure 1: The Knob.

The knob is capable of more than 100 settings per second, but most control system servers cannot accept such a setting rate, so the knob implements a configurable throttling period by default 500 ms. It can be configured to set values only touching the central button; but the main task is a continuous flux of settings. The user can change the scale of the inner wheel (as well as almost all colors and sizes are configurable). There is an optional chart, a reset to initial value button and a status indicator. The inner wheel scale can be adjusted by pressing the plus or minus green buttons on the bottom. In this case the external wheel disappears until the original scale is restored.

The Dodecagon

The "Dodecagon" (Fig. 2) is an application in which each of the 12 sections can be selected. Users can switch ON, OFF and standby High Voltage power supply and/or Feedbacks and set a value using the knob component.

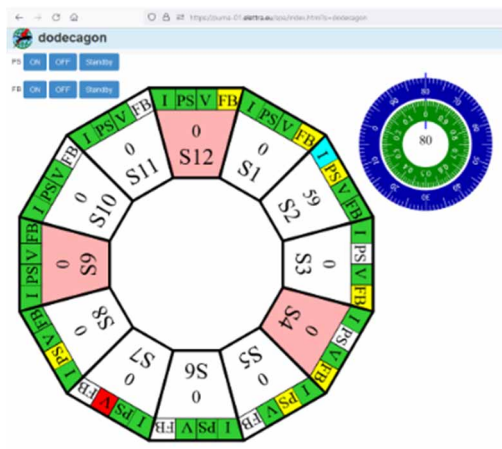

Figure 2: The Dodecagon.

* lucio.zambon@elettra.eu

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP036

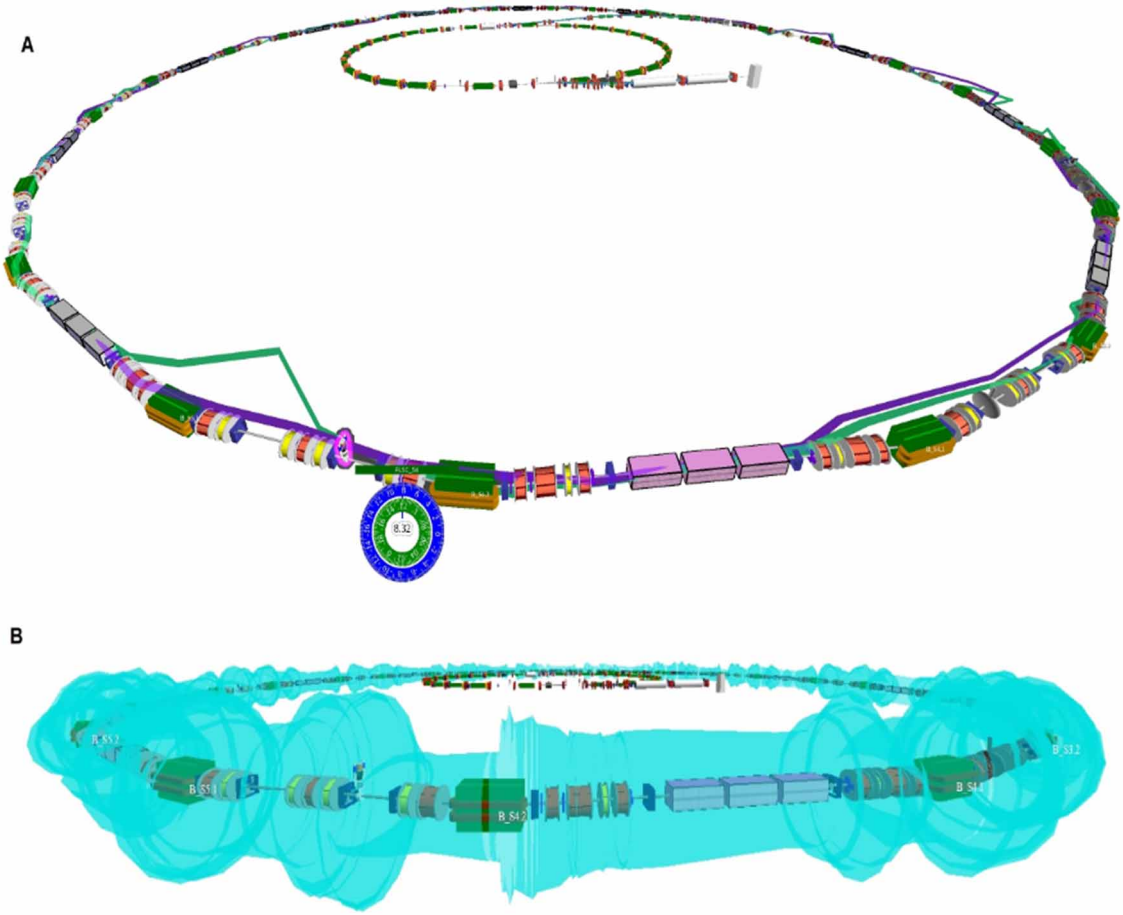

Figure 3: 3D model of Elettra. A: with beam position. B: with envelope.

3D MODELS

Three.js "is a cross-browser JavaScript library and application programming interface (API) used to create and display animated 3D computer graphics in a web browser using WebGL." (Wikipedia)

WebGL basically draws triangles in a 3D space efficiently using also GPU (Graphic Processing Unit) if available. Three.js requires Chromium or Firefox.

Three.js was used to construct PAnTHer a 3D model of a Particle Accelerator on THree.js (Fig. 3).

Our model is loaded from a JSON (JavaScript Object Notation) lattice file. A few lattice files have been developed starting from some Matlab (R) m-files. In these files we also inserted some hooks to the control system in order to insert some dynamic visualizations.

JSON files are generated programmatically by a PHP script which can put in evidence the differences between an already generated file (old file) and the result of the generation process using the actual m-files (new file)

The JSON file is subdivided in facilities (for example storage ring or booster) and each facility is an array of straight sections, normally beginning and ending in a bending magnet. All other components are placed at a certain distance from the beginning of the straight section.

A three Dimensional model requires some basic concepts:

a *scene* which contains all visible objects, one or more *cameras* to observe all visible objects. A *mash* is composed of a material and a geometry.

There are a few predefined *materials*, the main parameters of a material is the color and transparency. Color can be substituted by a *texture* for each face of the geometry. We used very small texture files for coils and undulators. There are several predefined geometries, the simplest is a box.

A *geometry* can be defined from scratch, it was necessary only in order to build a tube with a variable elliptical section in which also the ratio between elliptical axis is variable (so the shape of ellipses is variable).

The *frustum* is the area visible; from a camera only a cone is visible, but for optimization, objects too close and too far are not rendered. We also implemented a double frustum that is a different threshold in order to visualize labels (which are 2D objects) attached to 3D objects.

We implemented about 35 components (Fig. 4), mainly magnets among them: quadrupoles, sextupoles, bpm, fluorescent screens, some of them are trivial (a box) or variants of other components. All components are stored in a folder and there is a synoptic page for developers.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP036

Figure 4: Components.

No accelerator machine requires all components, most require a restricted subset of components. This is the motivation for grouping components in a bundle, that is a single JavaScript file containing all components required for a certain accelerator. Each bundle is automatically generated by a PHP script which detects the necessary components from the JSON lattice file and it is stored in the same folder together with a preview image of each component. Some components are clearly miming the reality (Fig.5) others are more symbolic.

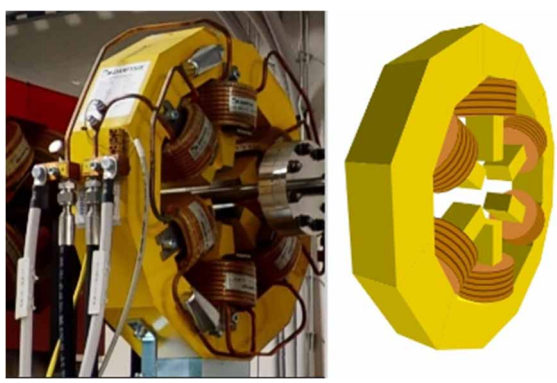

Figure 5: Booster sextupole.

For efficiency reason we implemented 3 sets of components: *normal*, *fast* and *plus*, the fast series is extremely symbolic in order to consume only a very limited amount of resources; the normal series is made of what we believe is the optimal compromise between resources used and "beauty" of the rendering; the plus version is aimed to be as beautiful as possible, but requires the most powerful GPU, graphical boards and several Gigabytes of memory, by now it is only a draft, we moved in this series the RoundedBoxGeometry (that is a box with rounded edges) when we realized that it was very memory consuming. This was the result of an analysis performed when we detected a clear slowing down of normal visualization which didn't affect the fast one.

We created a super bundle which included both normal and fast bundles and we added the possibility to pick the normal or fast version of each component independently. For each setup we measured the memory used. Although the browser doesn't release the memory immediately when it isn't necessary any more, it was clear that the components using the RoundedBoxGeometry increased the request of memory between 1000% and 2000%; so we decided to move it in the plus series.

It is possible to put in evidence one or more components reducing the size of all other components. The ratio of reduction can be adjusted dynamically (also the browser address bar is updated instantly). The result is a specialized synoptic of a given component.

Settings can be applied with a popup knob (the same SVG component described previously)

On top right corner there is a configuration menu implemented thorough lil-gui library [5], which is already included in Three.js, but it was extended adding two features: small icons used as buttons and the knob described above (Fig. 6).

Figure 6: Configuration Menu.

The PAnTHer development is aimed at visualizing live data from the machine or from simulators. Clicking or tapping on almost all components opens a popup displaying the main parameters of the object (for example a current or a position). In some cases there is a link to an external web application controlling that device.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP036

The component clicked is outlined by a blinking gradient highlight around the 2D form of the object (Fig. 3).

There is a component search engine with autocompletion suggested, based on JQuery UI autocomplete widget.

The most interesting features are 3 synoptic views: envelope, BPM (Beam Position Monitor) and BLM (Beam Loss Monitor)

Envelope (Fig. 3) is a transparent tube proportional to the statistical dimension of the beam magnified by a 10 million factor; envelope data is taken from a simulator; it would be also possible to zoom inside the vacuum chamber and find the beam in real dimension, but in this case it would be visible only in a few points at a time; instead with the magnification a synoptic view is possible.

BLMs data is displayed as a histogram with values on both sides of straight sections. All numeric data is displayed in a side flexbox table and the length of the histogram bars are calculated with a saturated logarithmic formula (Fig. 7).

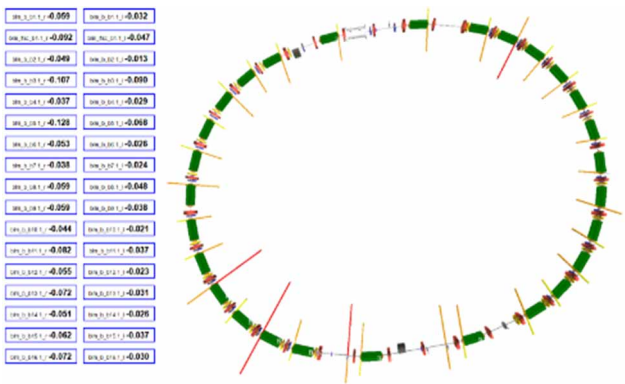

Figure 7: Beam Loss Monitors.

BPMs are the source of the data used to show the trajectory of the beam magnified by a factor 10M; in case of failure one or more BPMs can be excluded from the visualization using the skip attribute in the lattice file. BPM trajectory can be saved with a name (for example 2GeV or 2.4GeV) and can be used as a reference (with a different color) visualized together with the actual trajectory (Fig. 3).

CONCLUSION

The interaction with the knob component on a touch screen is considered by some users more similar to the interaction with a real knob than to the equivalent mouse interaction.

Until now PAnTHer received both enthusiastic and skeptical feedback. At first glance it is quite surprising; but the look and feel of 3D isn't appreciated by all users.

Anyway PAnTHer demonstrated that even very complex web applications may be as performing as embedded applications and that the times of development, debugging and updating of web applications may be considerably shorter than some embedded applications.

A live demo is at https://luciozambon.altervista.org/app/panther.php, source code are available at https://gitlab.elettra.eu/puma/client/web/-/tree/master/panther/

ACKNOWLEDGEMENTS

We acknowledge Giulio Gaio for dozens of ideas; Francesco Tripaldi and Enzo Benfatto for some very useful suggestions. Luca Sturari, Claudio Scafuri and Stefano Krecic for several lattice and mechanical models.

A special acknowledgement to Roberto Marizza, now retired, who many years ago pointed out the aeronautical industry idea of gauges keeping the default lancet position fixed on top center and having the gear to move according to the situation.

REFERENCES

[1] Hammer.js, https://hammerjs.github.io

[2] Three.js, https://threejs.org

[3] G. Strangolino, and L. Zambon, "Canone 3: a new service and development framework for the web and platform independent applications, in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 1023-1028. doi:10.18429/JACoW-ICALEPCS2021-FRAR02

[4] SVG, https://developer.mozilla.org/en-US/docs/Web/SVG

[5] Lil-gui, https://lil-gui.georgealways.com