Report on html-css-svg-etc. insertion into webGL
Rudolph August 2016

The task of inserting html/css/svg/etc (hereafter referred to simply as 'html') into webGL as a 'first-class' media type (including the full interactivity of pure html, and able to communicate seamlessly with webGL objects) has significant difficulties, multiple possible solution paths, and possible additional difficulties not considered here related to unknown specifics of the sensory-ware software interface such as possible event handling complications due to stereo rendering such as required in VR.

After considerable but not comprehensive research, it seems that possible solutions break into two categories, WebGL-only, and hybrid CSS3D/WebGL. (Most of the examples were from techniques found on the web and modified, and some were synthesized)

Let's consider first webGL-only rendering of html. While this category has the virtue of simplicity, it can only render the graphic 'appearance' of html. The critical factor of interactivity via pointer events is lost and must be re-created by additional means within webGL.

The best method of importing html into webGL is as a texture map, and great programming facility is gained by mapping the html 'image' to a canvas '2d' context and then texturing a webGL canvas '3d' context object surface.

First, these canvases can have bi-directional synchronisation and communication as seen in this pair of examples:

dynamic_texture_hud.html - this example tracks the rotation of a 3D-webGL cube and writes a dynamic text report to a 2d-canvas HUD (head's-up display). There are some annoyances - the immediate mode graphics of the 2d-canvas requires clearing the display and re-writing the complete text-graphics, and informing the webGL that the texture needs updating in each frame. The significance is 3D -> 2D synchronization and communication.

dynamic_date_texture - this example creates a dynamic 2d-canvas which displays the present date in ms. since Jan 1 1970. This canvas is used as a texture on each face of a 3D cube in webGL. This text-fill image is mapped dynamically on each cube face so the texture stays in sync with the 2d-canvas source. The significance is 2D -> 3D synchronization and communication.

These first two examples did not even exercise html so we move on to an interesting module I used several years ago when I was experimenting with a system I call 'vasarely' (after the painter Viktor Vasarely) for harvesting the 2d-canvas imageBuffer and using the pixel values to create small svg objects corresponding to the original image. Here is a sort of example:

medeterrenianDaisies.jpg - the analyzed image

vasarely.png - the png-screenshot of the generated svg-image (with a little pre-compositing with a second flower image)

Why show a vasarely example? Well I suggest that a variation of vasarely can be used to elegantly and systematically re-create interactivity in the 'dead' texture maps. Vasarely (sub)samples the imageBuffer and generates an svg

object (disk as in the example) having related (average) pixel color value. Imagine instead if we encode in the image representation of the html some 'coded' trigger via pixel color values or some essentially invisible delta-color value detectable only via vasarely code. Then at that region we generate a transparent webGL quad or shape which serves as the receiver of pointer events for the underlying texture mapped region of the html. It would be simple to synthesize simple pointer-event interaction. It might be harder to pass on keyboard text events, but it is plausible also. The event flow would be webGL event collection to 2d-canvas and then back to webGL texture and webGL renderer into texture map and vasarely analysis.

What is called for is an MR shape/color-code semantics standard across all MR-UI instances. A standard clickable shape would be nice so colorblind users can immediately know what is interactive. However the color is critical not only for color-seers, but for vasarely. Since all html is rendered as an image, the exact pixels and RGBA values assigned can be completely controlled to trigger precise vasarely-like behaviors for auto-generating dynamic interactive transparent wegGL sensors placed over the top of the 'dead' html and able to intercept pointer events in order to simulate completely interactive HTML in webGL. I would add at least one more dimension to the shape/color-code semantics - 'reaction' - I think some visual response to successful 'click' be synthesized - such as the water ripple effect seen in the example ripples.html. Independent of the html->webGL problem space a uniform shape-colorcode-reaction semantics is valuable to develop for MR. The most difficult part of the process is dynamic analysis of arbitrary pages to determine regions which generate events, and the associated event-handlers to invoke from the transparent webGL 'covers'. Of course this dynamic analysis only occurs in the case of an arbitrary iframe 'from the wild'. Magic Leap interafces, or interfaces prepared by third parties using guidelines and requirements for identifying 'hotspots' would be fairly easy to accommodate in this 'shape-colorcode-reaction' semantics system.

The motivation for the vasarely discussion is a module 'html2canvas' which faithfully maps html graphics to canvas graphics (with some inessential restrictions - no flash, applets, etc.) Again, the significance is that the 2d-canvas graphics can be used to texture map webGL objects and to trigger vasarely interaction.

Here are two examples - one which maps some html to a new canvas:
create_canvas.html
and the other which maps some html to an existing canvas:
existing_canvas.html

One further step is needed - what is the best means of producing the 2d-canvas texture maps? I believe that it is pixi.js which uses webGL to produce 2d graphics for 3d textures, but via webGL, which allows the use of shaders for visual effects. A system called 'html-gl' produces nice 2d pixi html graphics. (https://github.com/PixelsCommander/HTML-GL)
It would be nice if we could use pixi for 2d webGL and webGl for 3d but these two webGL renderers conflict and do not allow correct z-sorting, and possibly other problems. In any case pixi.js can be used to produce beautiful 2d textures utilizing webGL shaders not available to 2d-canvas.

Just to prove that pixi-webGL graphics can be used for webGL textures here is an example:

pixi.html

Now, let's consider the second category of solutions - using the CSS3DRenderer in parallel with the WebGLREnderer for rendering html as a 'first-class' media type in webGL. The main idea here is that rendering in css3D means that html interactivity can be preserved, as long as events pass through the webGL layer. The method of pass-through is the key.

Here is a first example of co-existing CSS3DRender and WebGLRenderer. Note by camera examination that correct z-order is maintained:
css3d_webGL.html

The next pair of examples is key for the consideration of simultaneous renderers. The first example correctly renders two webGL cubes, one in front of, and one behind, the css3D plane-iframe (use the camera controls to rotate). However, in this case the webgl context consumes all events and there is no possibility for interactivity with the css3D iframe, so the website is static and 'dead'.
css3d_inb_webGL.html

However, if pointer-events are turned off for the webGL canvas layer, the css3D-rendered iframe responds exactly like the pure html it is, and as in my email you can scroll down and read the artcle on 'the new medium of interactive virtual worlds', or look at the css3D examples, and other media examples on the i3Dmedia.org site:
css3d_interactive_inb_webGL.html

Of course turning off pointer-events for the webGL canvas is not a real solution. However, I think if the webGL object-picking code does not detect a webGL object intercepting the point ray, the event could be sent to the parent element, which as seen in the examples, is the CSS domElement div, parent to the child webGL domElement canvas, a necessary relationship to achieve proper z-sorting independent of the interactivity problem. Further understanding of the object-picking implementation is probably required, and further understanding of how the two renderers and their domElements interact is also required.

Two further specific iframe examples involve google maps. In the first case a css3D plane hosts the google iframe. The algorithm for forming the geometry of iframe planes allows complete freedom to produce a geometric set of planes corresponding to any threejs geometry presented, but I think a single plane is preferable to a 'sphere' of 16 planes for example. In any case the spherical camera interface is clumsy - simple is best.
However, maybe for some other purpose this 3D interface could be useful:
google-maps-faces.html (plane-1)
google-maps-faces-sphere.html (sphere-16)

Finally, a simple example shows that the CSS3DRenderer can render interactive DOM of all sorts including svg and an example textarea which is editable:
svg.html

In conclusion, until knowing specifics of the sensory-ware interface, and without further work on webGL and DOM pointer-event handling and 'bubbling',

it's hard to say that a single solution path stands out as the most promising. My intuition is that the event 'wiring' and handling for the hybrid case could be solved and therefore allow 'real' html in the CSS3DRenderer within the webGL space. The WebGLRenderer-only solution requires replacing the html-DOM events system with an equivalent webGL system. Maybe ultimately handling all events in webGL will prove to be simpler.