**VR-WEBB CLUSTER**

**Features and development –**

* **Star Rendering –**
* **Faster loading of stars –** Usage of BufferGeometry class was considered as a good solution. (**Not Implemented**).

**Links:**

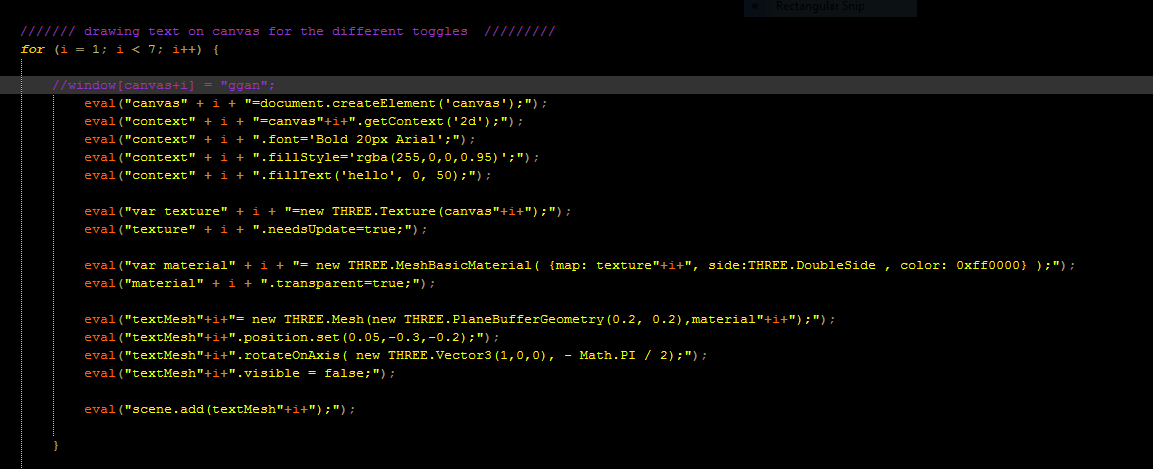
* + **Spec -** [**http://threejs.org/docs/#Reference/Core/BufferGeometry**](http://threejs.org/docs/#Reference/Core/BufferGeometry)
  + **Example:** <http://threejs.org/examples/#webgl_buffergeometry_custom_attributes_particles>
* **Star appearance –** 
  + Texture Loaderclass object from ThreeJS was used to load an image texture on the material of the stars**. (Implemented).**

**Points to consider:**

* + - **Transparent images –** Transparent images enabled transparency of stars but did not map image texture. So, in order to have stars rendered with the texture of the image non-transparent images with a black background were used to render stars
    - **Size Attenuation –** Size Attenuation parameter of the material applied to the stars enabled the stars that were closer to the camera to look bigger and the stars that were far off to look smaller proportional to their distance from the camera. This made the visible set of stars to disappear as those were comparatively further away than the stars from the cluster. The size attenuation parameter is a Boolean and can be easily turned off and on to render stars with size relative to their distance from the scene. This was enabled in the code to make the scene look less cluttered.
    - **Size-** The size of stars became an issue because of their appearance. After applying an image texture the size needed to be increased to greater values than before to achieve visibility. This was done by trial and error.
    - **Scale –** A combination of the two functions- **setScale() and multiplyScalar(),** was tried to achieve complete visibility of both sets of stars after the setting of Size Attenuation property. This did not work as a feasible solution and so was discarded.
    - **Overlapping of star positions with objects on the screen -** The properties - **depthTest** and **renderer. sortObjects** were enabled to ensure the render order of the stars and the objects on the scene which also was a feasible approach to the elimination of the overlapping of stars with other scene objects.
* **Menu Icons –** Menu Icons were rendered as image textures on box geometries (with no depth values).
* **Skybox –** The skybox was rendered as a huge scaled ThreeJS Mesh object. Scaling was done by trial and error.
* **Text –** Text in a 3D scene could be rendered in one of the following four ways –
  + In an HTML5 canvas element as a texture on a 3D mesh
  + ThreeJS rendered 3d Text using the native Three fontface librarary
  + Pre-printed text on images and using those images to display relevant info
  + Sprite text labels as shown here-[**https://stemkoski.github.io/Three.js/Sprite-Text-Labels.html**](https://stemkoski.github.io/Three.js/Sprite-Text-Labels.html)

Text in the scene was rendered using the first way and was toggled between states similar to any regular Three JS mesh object using the ‘visible’ flag.

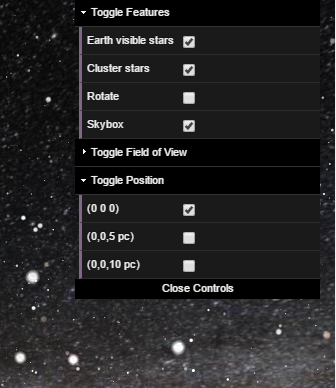
Following is a snapshot of the code snippet:



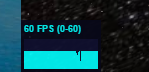
* **Remainder of the scene** - ThreeJS was a key player in rendering the remainder of the scene and examples on the official Threejs webpage were referred to help incorporate features in the application.

**Technologies used –**

* **Three.js -** Three.js allows the creation of GPU-accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins. This is possible thanks to the advent of WebGL. High-level libraries such as Three.js or GLGE, SceneJS, PhiloGL or a number of other libraries make it possible to author complex 3D computer animations that display in the browser without the effort required for a traditional standalone application or a plugin. (Source: <https://en.wikipedia.org/wiki/Three.js>)
* **WebVR –** WebVR is an experimental Javascript API that provides access to Virtual Reality devices, such as the Oculus Rift or Google Cardboard, in your browser. This was used to facilitate mobile device detection and virtual reality modes for the application and has the potential to be used for multiple VR headsets besides the Google Cardboard. The WebVR boilerplate was used as a foundation as it takes care of features such as VR controls(**VRControls.js**) such as enter and exit mode controls, effects for VR (**VREffects.js**) such as stereoscopy and support for other headset devices (**webvr-polyfill**).
* **datGUI -** datGUI is a lightweight graphical user interface for changing variables in JavaScript (**Source:** <http://workshop.chromeexperiments.com/>). In the context of a ThreeJS scene these could be regular ThreeJS mesh visibility toggles or attributes such as speed of any moving particles, size of particles, physics of a scene, or other rendering attributes. The datGUI panel was incorporated to allow for user-controls on a desktop for the application. Following is a snapshot from the application –



* **stats.js -** A JavaScript performance widget. Stats.js adds a simple info box to your page displaying frames per second (FPS) or, with a mouse click, the milliseconds required to render the frame. (**Source:** <http://workshop.chromeexperiments.com/>) This was also incorporated as a part of the scene. Following is a snapshot:



* **Reticulum.js -** Reticulum attempts to follow Google's interactive pattern for the display reticle. It creates the illusion of depth by projecting spatially onto targeted objects while maintaining a fixed size so that it is easy to see at all times.

**Features:**

* Avoids double vision and depth issues by projecting spatially onto targeted objects
* Gaze and click events for targeted objects onGazeOver, onGazeOut, onGazeLong and onGazeClick
* Set different fuze durations for targeted objects
* Built in fuse support
* Display the reticle only when the camera can see a targeted object
* Works in the browser with Three.js (r73) (**Source:** <https://github.com/skezo/Reticulum>)

Reticle(initial)

datGUI panel

**APPLICATION SNAPSHOTS**



Stars and skybox

Fullscreen toggle

statsJS panel

Text toggle for actions

Reference Grid



Active Reticle

Toggle Action buttons for Visible stars, Cluster stars, Rotation, Field of View and Position

