During my first several months as a research assistant for the Blackfoot Research Project, I was tasked with two primary goals. First, I was asked to explore means of supporting and displaying 3D scans of Blackfoot artifacts on the web. This was completely new for me at the time, and I had very little prior experience with 3D objects in general, let alone supporting them on websites. As a recent graduate with a bachelor’s degree in computer science, I was keen to tackle this new challenge.

I started my search with the popular JavaScript library three.js. Three.js gave developers a lot of freedom in crafting and hosting web-based 3D objects. The issue with three.js was that it came with a lot of unnecessary features, and by extension, increased complexity. Firstly, I knew that we did not need to craft our 3D models declaratively, because we already had high quality scans. All we truly needed was a means of displaying those models on the web, and allowing users to interact with them. Secondly, each and every model that gets rendered by three.js would require it’s own JavaScript code in order to render it. Because each artifact is different, we would potentially require hundreds (if not thousands) of lines of code in order to support each and every Blackfoot artifact. Additionally, each way these artifacts would be displayed would further increase the workload for rendering these artifacts. What took just shy of one-hundred lines of code to render a single artifact would need to be repeated upwards of thirty times. Finally, I knew that the desire to add additional artifacts to the collection was strong, and adding additional artifacts to the collection would mean that new code would have to be written in order to support the new artifacts. I knew there had to be an easier way.

Early in my search for a better way to support models on the web, I discovered a web-component being developed by Google called Model-Viewer. Model-Viewer is a light-weight 3D modeling framework built on the three.js suite. This framework takes a lot of the backend workload out of the equation, allowing for self-contained models which do not require the same care and meticulous hassle that comes from using three.js, while including no performance penalties. I thoroughly tested the features of the web-component, and studied how it worked, and whether it would satisfy the needs of the Blackfoot Research Project. I soon established that it would, given that most of the features I tested worked perfectly, and due to the nature of the web component, it alleviated many of the issues that would be included with three.js in its base form. Practically every 3D modelling functionality I figured we would need was already included in Model-Viewer, including rotation, zooming, animation, hotspots and more.

When I first discovered Model-Viewer, it was still in development, and I kept a close watch on the development repository to verify that bugs were indeed being fixed, and promised features were being added, like hotspots. In the time I dedicated to studying Model-Viewer, many improvements and features were added, and support for the platform is only improving. At the time of writing, this tool is still being actively developed by Google, and the 1.0 version of the platform has yet to be fully released.

The second area of research I was involved with was the Google Maps platform. In order to convey locational data related to the artifacts in a visual way, we required a platform which gave us the flexibility we needed. Additionally, Google Maps is one of the more popular maps platforms out there, which means that users of the site will likely already be familiar with Google Maps, and prior knowledge will reduce the learning curve associated with using the maps on the website. Google Maps suited our needs perfectly in this regard.

Prior to my research into the Google maps Platform, I had no experience with Google Maps as a developer, only as a user. I took several tutorials on using the Google Maps JavaScript API, and quickly began experimenting with the features that Google provides. I learned how to place markers, create info windows, and customize the aesthetic of the map through styling. Due to how important map aesthetics is for the project, I really wanted to identify what was possible in terms of customization, and what was not. While Google Maps does offer quite a lot customization options, there are certain portions that offer very little in terms of customization. For example, Info Windows themselves offer virtually nothing in terms of aesthetic customization, though their content is quite flexible.

Another area of exploration within Google Maps was how expandable it is, and how straight forward it is to add to it later on. Due to additional artifacts being potentially added to the collection later, it was crucial that the map could be easily manipulated later. I created several basic functions which were aimed at facilitating additions to the maps, such as easily adding markers, etc. This research translated reasonably well to the React version of Google Maps which is currently being used for the development of the site. Due to the inherent differences between Google Maps’ JavaScript API, and the library used within React, there are several features presented on this demo site that may not be possible. However, the majority of features are included, and the ease of adding to, and manipulating the maps in React is a valuable tradeoff.

React is able to handle the expansion of the site extremely well, and adding additional artifacts to the site in general, and by extension, Google Maps is much easier, requiring no changes or additions to be made to the core code of the site.