From Theory to Practice: Resolving ArcGIS Integration in Deck.gl

Jared Scarr

Abstract

This study documents the involvement of a team of university students in a collaborative program between CodeDay and a university. The team, comprised of three members and guided by a mentor, embarked on a journey to contribute to an open-source project hosted on GitHub. Despite possessing approximately three years of experience in software development, the team members had not previously engaged in open-source contributions. This paper aims to elucidate the critical insights and takeaways from this experience, offering guidance and potential strategies for other first-time contributors to open-source projects.

Introduction

Deck.gl is a powerful library designed for the visualization of extensive geospatial datasets and maps utilizing WebGL technology. The typical users of this framework are individuals or teams, often occupying roles as developers or data scientists. For a comprehensive selection of example reports and applications, refer to the Deck.gl showcase.

Deck.gl uses the following terms:

- **Layer**: A fundamental component of Deck.gl that can be superimposed on a map or other layers to enhance data visualization.
- **Texture**: An image within the GPU that corresponds to actual images, colors, etc., used for rendering purposes.
- **Frame Buffer Object (FBO)**: An OpenGL extension that facilitates the rendering of a texture.
- **Binding**: The process of connecting a texture to the renderer, enabling the rendering process.

An example of Deck.gl is presented in Figure 1. The ArcGIS base map is presented with two additional layers superimposed. The first layer consists of geo-JSON data, representing the locations of various airports, which are highlighted using magenta circles. The second layer comprises arced lines that visually connect the airports, illustrating their interconnections.

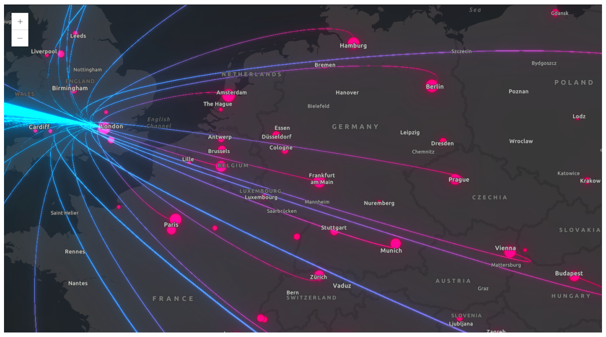

Figure 1: ArcGis base map is displayed with two layers.

Figure 2 illustrates a basic diagram of the operational workflow. Deck.gl offers a pure JavaScript API and also supports integration with React. Initially, a base map is selected. Subsequently, one of the core features, the Deck component, is instantiated and assigned layers. These layers are rendered on the screen, superimposed over the base map.

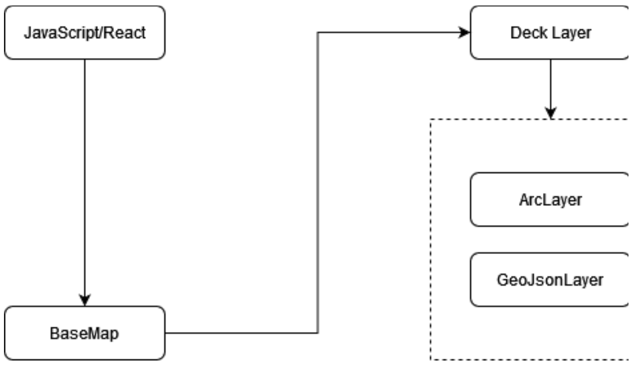

Figure 2

The focus of this case study was to address the broken ArcGIS integration, which prevented developers from rendering DeckGL data on top of the ArcGIS map. Despite the documentation appearing accurate, the library itself was malfunctioning. The initial task was to isolate the specific segment of the code where the issue resided. Two main areas were identified as crucial: the example application provided for guidance and the ArcGIS module.

Methodology and Solution

The initial strategy involved identifying error messages in the console, decomposing them into sub-problems, and searching for similar examples within the existing codebase to use as references. However, it was soon discovered that none of the error messages provided immediate insights. Additionally, a portion of the code was inaccessible, operating as a closed-box, which posed a significant challenge for debugging. Consequently, the initial strategy proved ineffective, raising the question of how to debug a system that could not be stepped through directly.

When commencing this project, the team lacked experience with shaders, graphics, or GPU-related development. Fortunately, their mentor possessed substantial expertise in these areas and provided invaluable assistance, including an article explaining a shader, which proved to be highly beneficial.

The team was accustomed to a development environment where all code was visible and could be stepped through using a debugger. In contrast, this system presented opaque sections, necessitating a different approach. Diagrams were drawn, and assumptions were formulated regarding the system's functionality. The team then implemented code to validate or refute these assumptions. Each change was made incrementally, followed by a page reload to document the outcomes. Whenever they encountered obstacles, they sought guidance from the project maintainers, the Slack community, and their mentor.

To identify the code sections requiring updates, the team traced each method call and pinpointed the specific code segments responsible for rendering to the screen.

The final solution involved creating a pipeline of components that could be isolated to diagnose issues effectively. This systematic approach enabled the team to identify and address the broken ArcGIS integration, ultimately restoring the ability to render DeckGL data on top of the ArcGIS map.

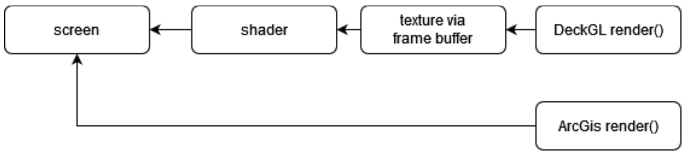

Figure 3

In the intended process (Figure 3), ArcGIS should render directly to the screen while DeckGL renders to a texture, which is then displayed on the screen. The investigation began with the shader itself. The initial step involved removing both the render and the texture Frame Buffer Object (FBO) to determine if a gradient could be drawn directly on the screen (Figure 4). The failure of this attempt confirmed that the issue resided within the shader.

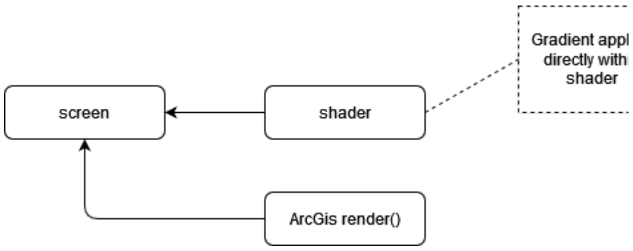

Figure 4

Through a process of trial and error, it was determined that the geometry of the shader required an update. Once this adjustment was made, the gradient successfully rendered, confirming that the shader was capable of drawing to the screen. This validation step was crucial in diagnosing and resolving the underlying issue within the shader. (Figure 5)

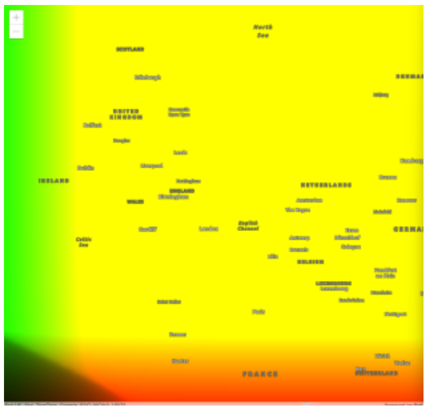

Figure 5

The subsequent logical step was to examine the communication between the Deck instance at the pipeline's other end and the shader. To do this, both the FBO and DeckGL render were removed and substituted with a hard-coded image to determine if it would render (Figure 6).

If the image rendered successfully, this would indicate that the shader was functioning correctly. Conversely, if the image did not render, it would suggest that the shader was still defective, and the DeckGL render might also be problematic. The image did render successfully (Figure 7), confirming that the shader was operating correctly.

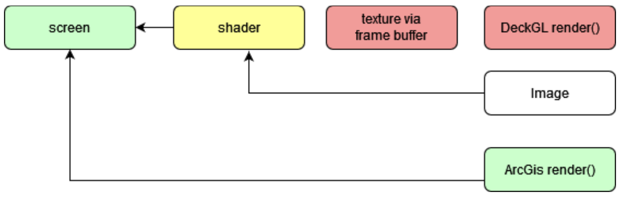

Figure 6

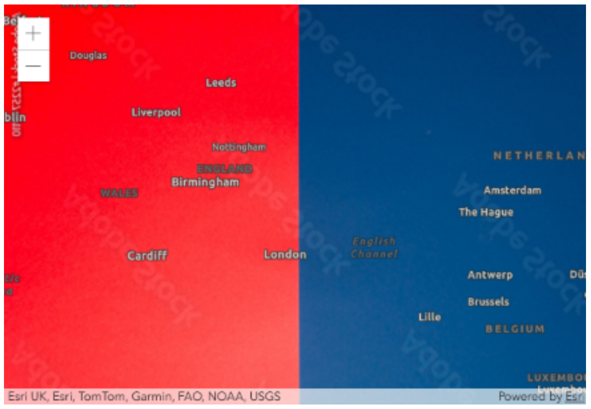

Figure 7

This verification confirmed that the shader could render correctly when supplied with a properly formed and bound image texture. The next step was to examine the DeckGL render. It was identified that the binding of the texture to the DeckGL instance required updating with newly required attributes. Upon discovering and implementing these necessary updates, the DeckGL instance successfully rendered to the shader (see Figure 8).

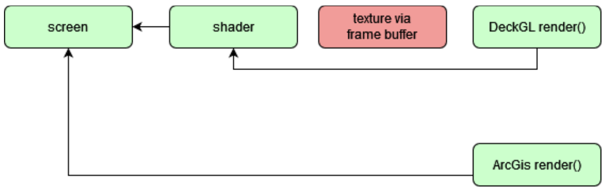

Figure 8

The final component to address was the frame buffer object (FBO). It was necessary to reconstruct the FBO and the texture with the correct parameters and ensure the texture was properly bound. Once the correct combination of attributes was identified, the map with the proper layers rendered successfully, and the sample image was removed (see Figure 1). This functionality was verified by adding a video in the discussion, demonstrating the example working in the browser. The maintainers accepted the pull request, marking the team's first successful open-source contribution.

Conclusion

One of the less obvious but crucial lessons learned from this experience was the importance of meticulously tracking progress and knowledge gained. With the numerous rabbit holes, code changes, side-quests, and validated or invalidated assumptions encountered along the way, it is easy to lose track of weekly activities. Maintaining detailed documentation proved invaluable, particularly for writing comprehensive articles or quality pull requests that explain the changes and their necessity.

Participation in this program was a highly rewarding experience, offering substantial learning opportunities both from the process and from the mentor. While not everyone may have access to such guidance, it is encouraged to seek out open-source projects for potential contributions. Look for issues labeled “good first issue,” join their Slack or similar communities, and thoroughly read their contribution guidelines.