Process Book Data Visualization

Disaster Visualization

Zacharie Mizeret (270849), Andrija Kolić (336957)
Ahmed Reda Seghrouchni, Mehdi Berrada

June 2023

1 Path

1.1 Initial Concept

The initial inspiration for this project derived from a video created by Sebastien Lague, in which he developed a captivating video game enabling users to navigate a virtual world in a plane freely. We were particularly drawn to the immersive and rewarding nature of the game, as it allowed users to explore the globe in an intimate and unrestricted manner. Moreover, transforming the learning experience into a game-like format tends to enhance user engagement and attention.

Building upon this initial concept, we made a collective decision to adopt the idea of employing a small plane traversing the globe as a means to present global data visually. One of our team members stumbled upon a compelling dataset documenting natural disasters that have occurred worldwide over the past century. Consequently, we chose to leverage this dataset to create a visualization that enables users to navigate the globe and observe the locations of various disasters.

Once we established this idea and received team approval, we faced the task of resolving some essential details. Primarily, we needed to determine how users would be able to navigate through different years. Given that the game takes place in a three-dimensional world, we recognized the need to incorporate a fourth dimension in a user-friendly and enjoyable manner. To address this, we decided to implement a slider positioned at the top of the page. This slider allows users to visualize the current year being displayed and easily navigate to their desired year.

During the data exploration process, we encountered a substantial volume of disasters, particularly for certain years. Presenting all of these events simultaneously could overwhelm users. Considering the diverse range of disaster types, we devised a feature that empowers users to select which specific disasters they wish to view. This approach serves two purposes: enabling users to focus on the types of disasters that interest them and reducing the number of displayed events on the globe, ensuring a more manageable and comprehensible visualization.

1.2 Data Processing

The most important part was ensuring we had usable spatial data for most of our dataset. Originally most of the spatial data was textual: country name and more specific locations. For the visualization we needed longitude and latitude coordinates, so we had to do some mapping, and for this we utilized the Nominatim geocoder service, using the interface of the geopy library.

Some progress was already made for Milestone 2, namely, we found coordinates for all of the countries in the dataset, which provided a very rough estimate of where the disaster occured. For

some disasters, this was sufficient, since they occured countrywide, but for most it wasn't good enough. Additionaly, using only the country coordinates would provide for a very underwhelming visualization. Similarly, some work was also done for Milestone 2 regarding decoding the coordinates for location data, but the results weren't that great.

At the starting point of Milestone 3, 55% of the data didn't have coordinates more precise than the respective country. The problem with previous attempts at decoding the location data was that a lot of the values are in a more complex format, either containing multiple locations, or containing perhaps excessive information that would just confuse the geocoder.

We conducted a brief manual analysis on samples of the dataset in order to figure out the best ways to parse the location data. This analysis resulted in us forming the following groups of the data:

1. A list of terms that individually look ready for geocoding. The terms are separated with characters such as ',' or ';'. Examples:
   - Susinga, Karawa (North Equateur), Baringa, Likukuma, Toenga, Dongila, Yakoma (Equateur)
   - Oklahoma, Missouri, Arkansas, Tennessee; Kansas, South Carolina
2. A list of terms that have words like 'district', 'city', 'province', 'state' or 'area' as the last word (referenced as *district words* from here on).
   - Guangdong Sheng province
   - Choïa District, Altai Republic (Sibérie)
   - San Sebastian city (Gipuzkoa district, País Vasco/Euskadi province)

   E.g. geocoding for 'Guangdong Sheng province' returns no results, while using 'Guangdong Sheng' results in the expected coordinates.
3. Descriptive terms.
   - near San Pedro Soloma
   - Border with Burkina Faso
   - North, Northeast
4. Weird looking values.
   - NO DATA: all provinces have been selected: Maradi, Niamey, Tahoua, Tillaberi, Zinder provinces
   - Town Of Tabuk Level 2 = Kalinga-Apayao (from GENAME)

Obviusly, the first step was using delimiter characters such as ',' and ';' to split the strings into individual terms. This helps with all 4 groups, since all of them can be lists, and are sometimes lists of terms that individually belong to different groups. We also decided to discard any information inside of parentheses, as it showed to be excessive for geocoding in our tests. A possible improvement could be made if this data was included, which would help in differentiating two specific cities/locations with the same name, but in different subdivisions of the country. These cases certainly represent a negligible part of the data, if present at all. Next, we decided to remove any *district words* present in terms, in order to solve the issue identified in group 2. The *district words* list was formed

	Country	Location	Smart Coords
693	Brazil	Parana	[{'Lat': -24.4842187, 'Lng': -51.8148872}]
7524	Viet Nam	Quang Binh, Quang Tri, Thua Thien - Hue, Quang Nam, Quang Ngai, Binh Dinh, Da Nang City provinces	[{'Lat': 17.509599, 'Lng': 106.4004452}, {'Lat': 16.7897806, 'Lng': 106.9797431}, {'Lat': 16.3480798, 'Lng': 107.5398913}, {'Lat': 15.5761698, 'Lng': 108.0527132}, {'Lat': 14.9953739, 'Lng': 108.691729}, {'Lat': 14.0779378, 'Lng': 108.9898798}, {'Lat': 16.068, 'Lng': 108.212}]
5074	United States	Kansas, Missouri	[{'Lat': 38.27312, 'Lng': -98.5821872}, {'Lat': 38.7604815, 'Lng': -92.5617875}]
13711	Myanmar	Lahe, Nanyun cities	[{'Lat': 26.3252881, 'Lng': 95.4423223}, {'Lat': 26.9798882, 'Lng': 96.1671674}]
12129	Switzerland	Juras Bernois, Alpes Bernoises area (Bern province) , Bas-Valais area (Valais province)	[{'Lat': 46.4324037, 'Lng': 7.725489508388815}, {'Lat': 46.2820488, 'Lng': 7.4826142}]

Figure 1: Sample of rows for which the described process was successful. Some rows have a decently long list of values, while for some not all terms were successfully found ('Juras Bernois' appears to not have been found).

	Country	Location
484	Gambia	Northwestern region
4380	Luxembourg	NaN
3481	United States	NaN
5531	Japan	NaN
3456	Philippines	NaN
9469	Fiji	Northern, Eastern, Central, Western provinces
14689	Oman	NaN
15589	Honduras	NaN
8309	Russian Federation	NaN
2588	Ethiopia	NaN

Figure 2: Sample of rows for which the described process was not successful. A lot of the values are missing, and the rest are comprised of descriptive words.

by counting words that appeared at the end of terms, and then showing the most frequent ones, manually removing words that should not be included. Next, we removed any descriptive words, in hope of helping with group 3, or at least removing unnecessary data from our geocode set. This list was compiled in a similar fashion to the *district words* list. The country information was retained through these transformations, allowing us to ensure that we don't get false positives that are in the incorrect country.

After performing geocoding on this set, we used the same process in order to merge the results with our original dataset, resulting in a new column named *Smart Coords* that contained a list of coordinates.

The result was that now we had very precise coordinates for 80% of the disasters, while for the remaining 20% the coordinates of their country could be used. More than half of the mentioned 20% had no location data to geocode. Another improvement was that we could now visualize events that covered multiple locations, since we a list of coordinates in the *Smart Coords* field.

We don't have any guarantees that this process resulted in coordinates that are 100% correct, there could have been some false positives. However, checking them all would be far too labor

intensive and make the automation meaningless, and from our tests the precision seems to be very good.

1.3 Implementation Choice

To effectively implement our visualization, we recognized the need for robust tools capable of presenting the extensive information we intended to convey. Traditional elements such as widgets, sliders, and buttons would prove inadequate for our purposes. Consequently, one of our team members introduced us to REGL—a functional abstraction of WebGL, which, in turn, serves as a web adaptation of the renowned rendering library OpenGL. Opting for REGL enabled us to leverage code reminiscent of that employed in video game development, facilitating the creation of dynamic scenes complete with realistic lighting simulations. Most notably, REGL offered a straightforward approach to rendering a three-dimensional scene onto a two-dimensional screen, bridging the gap between spatial depth and visual representation.

1.3.1 REGL Key Notions

In the subsequent section, it is important to provide an explanation of some key concepts regarding the functioning of REGL. Primarily, it is essential to grasp that every object within a scene is defined by three main components, simplified for clarity: a mesh, a vertex shader, and a fragment shader. These three elements collectively determine the appearance and characteristics of the objects portrayed in the scene.

Mesh
A mesh contains vital information about an object in the scene and is composed of four main components (simplified for clarity): vertices, faces, normals, and UV coordinates. Vertices represent points on the mesh, each defined by a three-dimensional coordinate (x, y, z). As most 3D objects are constructed using triangles, the faces of a mesh define these triangles by specifying a set of three vertices that form a triangle.

Normals refer to the vectors perpendicular to each triangular face. They play a crucial role in simulating lighting effects, providing information about the object's orientation and aiding in optimizations by determining which triangles should not be rendered if they face away from the viewer.

UV coordinates establish a mapping between the three-dimensional vertices and two-dimensional points. This mapping is particularly useful for texture sampling, allowing the application of textures or images onto the surface of a 3D shape. By assigning UV coordinates, we can map the shape onto a flat 2D image while preserving its original appearance and details.

Vertex Shader
The vertex shader plays a crucial role in WebGL/OpenGL programming, executing in parallel for each vertex. It serves as the initial step in the rendering process for an object, allowing us to interact with and modify vertex properties. Through the vertex shader, we can perform various computations to manipulate the position of each vertex as needed. Additionally, this program provides an opportunity to prepare and transmit any additional information required for the fragment shader.

Fragment Shader
The fragment shader is executed following the vertex shader and operates in parallel for each pixel on the screen. Its primary purpose is to determine the resulting color of each pixel based on the

information provided by the vertex shader. This shader enables us to perform various computations that contribute to realistic rendering, including the simulation of lighting effects such as shadows.

1.3.2 World

In order to explain the process behind achieving our visualization, it is essential to delve into the steps we took. The initial and arguably most significant aspect was the creation of the globe itself. This particular stage presented a challenge as we needed the mesh to possess two critical features: adjustable definition for efficient generation and a stable, well-wrapped UV mapping around the globe.

Initially, our intention was to utilize a pre-existing mesh created in Blender. However, we soon realized that if we wanted the ability to vary the level of detail, it would require generating numerous versions of the globe, leading to inefficient use of space and resources. Consequently, we made the decision to construct our own mesh by employing mathematical calculations to project two-dimensional (x, y) coordinates onto three-dimensional (x, y, z) coordinates. This approach not only facilitated the creation of the desired UV coordinates but also provided a straightforward solution to address the issue of texture sampling accuracy.

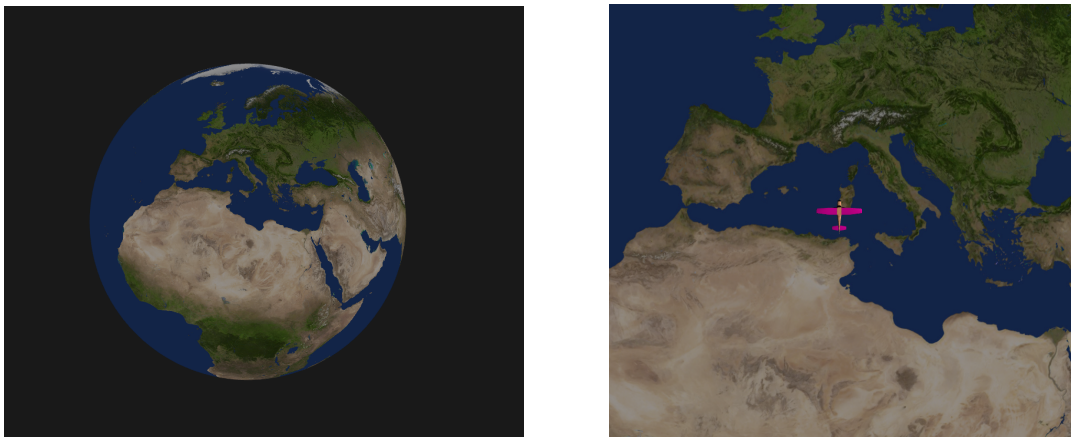
While some UV mappings utilize trigonometric functions to convert between 3D and 2D coordinates, we encountered limitations when working with edge case values of these functions in JavaScript. The precision of JS for such calculations proved less than ideal, resulting in highly unstable functions and inaccurate texture sampling during subsequent stages of the visualization process.

Once the object's coordinates were established relative to their respective origins, the subsequent task involved creating the necessary "camera" matrices. These matrices played a crucial role in converting object coordinates from their local space to the world space. Additionally, the world space coordinates were further transformed into camera coordinates. Setting up these matrices proved relatively straightforward, and with their implementation complete, our focus shifted towards constructing the rendering pipeline to display the globe on the screen.

To accomplish this, we employed a simple vertex shader. This shader was responsible for converting the globe's local coordinates to camera coordinates. Subsequently, the fragment shader was utilized to apply a large image of the world as a texture onto the surface of the globe. Initially, we attempted to incorporate shadows using normal mapping techniques. However, the results did not meet our expectations, and we opted to discard this approach. Hence, we will not delve further into the details of this unsuccessful endeavor. We can see the initial rendering we obtained in figure **??**.

Subsequently, our attention turned to incorporating the plane into the environment. To ensure that the plane remained consistently centered within the world, we assigned it a dedicated "camera" that ensured its accurate rendering. The plane mesh itself was sourced online, and we applied a custom texture on top of it. As our artistic abilities were limited, we opted for vibrant and playful colors to adorn the plane's surface.

In terms of the rendering pipeline, we converted the plane's local coordinates into its own camera's local coordinates. This transformation allowed us to accurately position and orient the plane within the scene. To enhance the visual appeal and provide a three-dimensional appearance, we employed lighting simulation techniques within the fragment shader. This enabled the generation of subtle shadows and added depth to the plane's rendering, contributing to a more realistic representation within the visualization. This stage of the visualization can be seen in figure 3b.



(a) Initial globe rendering.

(b) Globe with the plane.

Figure 3: Renderings at various stage of the development process

1.3.3 Controls

Once the objects were in place and the rendering pipeline was set up, the next step was to add controls to the visualization. To achieve this, we utilized event listeners that responded to the W, A, and D keys. These event listeners were responsible for modifying the camera matrix, which played a crucial role in converting global coordinates to camera coordinates.

During each frame, the matrices were updated to incorporate any necessary transformations. The updated matrices were then passed to the rendering pipeline, ensuring that the visual output accurately reflected the changes made through the control inputs. By continuously updating and applying the matrices, the controls seamlessly influenced the rendering process, providing an interactive and responsive user experience.

1.3.4 Disasters

In this stage, we proceeded to integrate the disasters onto the globe. After performing the necessary data cleanup, the events were loaded into the code, corresponding to the selected year (further explained in the following section). To visually represent each disaster, we created cube objects with colors corresponding to the specific type of catastrophe. By converting the longitude and latitude coordinates of each event into three-dimensional coordinates on the globe, we prepared the data for the rendering pipeline.

To enhance the visual depth and realism of the cubes, we incorporated lighting simulation techniques into their rendering. This additional touch added a sense of three-dimensionality, elevating the overall visual experience. However, during this process, we encountered a persistent issue known as z-fighting. When two cubes were positioned on top of or extremely close to each other, the rendering pipeline faced difficulty determining which object should appear on top. Consequently, this led to flickering between the two objects, as the rendering order fluctuated. Despite our efforts, we were unable to resolve this z-fighting problem completely.

During this stage, we encountered an obstacle concerning the representation of different types of disasters. Ideally, we aimed to have unique meshes for each disaster, allowing for visually distinct elements such as a wave-like structure for floods or other appropriate shapes. However, due to some

team members not fulfilling their responsibilities, we faced a setback in the development process, resulting in the inability to create these specialized objects. Consequently, we had to resort to using colored cubes as a substitute.

1.3.5 UI Elements

In the concluding phase, we integrated additional user interface (UI) elements to enhance user control over the displayed events on the globe and provide comprehensive information about the disasters. These additions aimed to facilitate a more interactive and informative experience for the users.

The first UI element introduced was a slider positioned at the top of the screen, prominently displaying the currently selected year for disaster visualization. Users could effortlessly slide this control to select their desired year, resulting in dynamic updates that instantly reflected the chosen timeframe. However, the implementation of this feature posed a significant challenge.

Initially, we encountered performance issues when attempting to instantiate numerous objects for each disaster during runtime, leading to excessive loading times and rendering the visualization impractical. To overcome this hurdle, we devised an optimization strategy. At the program's initiation, we pre-instantiated a substantial number of objects, representing all possible disasters. Consequently, when the user changed the selected year, these pre-existing objects were efficiently updated to reflect the specific disasters relevant to that year. This approach ensured instant updates and eliminated any loading time concerns, delivering a seamless and responsive user experience.

The second UI element incorporated was a checkbox form positioned at the bottom-right corner of the interface. This intuitive control allowed users to selectively choose which types of disasters they wished to visualize on the globe. To simplify the selection process, we employed a color-coded scheme where each checkbox corresponded to the color associated with a specific disaster type. This design served two essential purposes: it acted as a visual key, enabling users to easily identify the corresponding disaster type by its color, and it provided an intuitive means for users to control and monitor which types of disasters were displayed within the visualization.

Finally, we introduced an interactive feature that allowed users to gain deeper insights into specific disasters. By hovering the cursor over individual cubes representing the disasters, the corresponding cube would be highlighted, ensuring clearer identification. Additionally, a small text box would appear, presenting users with comprehensive information about the selected disaster. This information included details such as the disaster's location, the duration of its effect, the number of people affected, and the number of fatalities.

These UI enhancements not only empowered users with greater control over the displayed data but also enriched their understanding of the disasters by providing contextual information and promoting a more immersive exploration of the visualization. The final rendering of our visualization can be observed in figure 4.

2 Peer Assessment

Our project team consisted of four members: Zacharie Mizeret, Andrija Kolić, Ahmed Reda Seghrouchni, and Mehdi Berrada. Each member had specific responsibilities assigned to them, dividing the workload as follows:

Zacharie Mizeret took charge of developing the rendering pipeline and generating the globe. This involved implementing the necessary algorithms and techniques to render the visual elements effectively.

Andrija Kolić focused on data cleaning and loading. They handled acquiring the data, ensuring its quality through cleaning processes, and implementing efficient loading procedures.

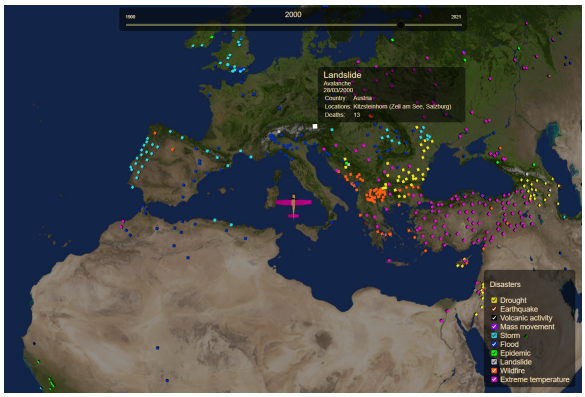


Figure 4: Renderings at various stage of the development process

Both Andrija Kolić and Zacharie Mizeret collaborated on designing and refining the user interface (UI) elements.

Ahmed Reda Seghrouchni and Mehdi Berrada were assigned the task of creating custom objects for the disasters. However, due to unforeseen circumstances, this task was not completed. As a result, alternative representations, such as colored cubes, were used to depict the different types of disasters.

Andrija Kolić and Zacharie Mizeret also took responsibility for documenting the project's progress by preparing the process book and writing the project report.

Although the workload distribution was initially planned to be balanced, the project faced challenges as some team members failed to join meetings or failed to complete their assigned tasks. Due to limited team resources and unforeseen challenges, the project was completed by two individuals, resulting in the need for certain shortcuts to ensure timely completion.