University of Crete – Department of Computer Science

Diploma Thesis

Supervisor (Professor):

Konstantinos Stefanidis

Advisor:

Emmanouil Zidianakis

Winter Semester 2021–2022

System for Visualization and Processing of Three-Dimensional Virtual Exhibitions in a Web Environment

Aldo Xhako (4052)

*This document is an **AI-assisted** English translation of the original Greek thesis.*

Abstract

The “Invisible Museum” is a project that aims to maximize the potential of Virtual Museums (VMs) through the use of cutting-edge technologies. Following the paradigm of the “new museology,” the project seeks to create highly collaborative and participatory Virtual Museums that provide personalized, interactive, and engaging user experiences through dynamically reconfigurable Virtual Exhibitions.

The objective of this diploma thesis is to analyze the process and usage of a system for the visualization and processing of three-dimensional virtual exhibitions, called “Exhibition Designer.” The Designer is a tool that allows users to create virtual exhibitions from scratch or even recreate representations of real-world exhibitions.

Table of Contents

Introduction

The Invisible Museum is a platform where users can upload their exhibits, create virtual exhibitions, and add narratives on top of them in order to provide a realistic museum experience for anyone with access to a web browser.

The scope of this undergraduate thesis covers the system for creating and editing the three-dimensional exhibitions hosted on this platform. Since every component of the main platform is hosted on the Web, this system had to be implemented in a way that allows seamless integration, ensuring smooth and uninterrupted use.

For this purpose, the A-Frame framework was chosen as the foundation. A-Frame is an open-source web framework for building virtual reality experiences. It is an Entity-Component System (ECS) built on top of Three.js, where developers can create 3D scenes and WebVR experiences using HTML.

While A-Frame proved ideal for setting up the virtual scene, the API it provided was too limited for the requirements of this project. The solution to this issue was the direct use of Three.js. Three.js is a cross-browser JavaScript library and application programming interface used to create and display animated three-dimensional computer graphics (CG) in a web browser using WebGL.

This library, along with the use of jQuery for communication with A-Frame elements, provides all the necessary APIs for implementing every functional component of the system for visualization and processing of three-dimensional virtual exhibitions.

Using the System

Designing the Exhibition Space

The exhibition editor is the part of the platform where the user gathers all the exhibits they have previously uploaded into a single space. Starting with a two-dimensional, tile-based room editor, the user can design the space that will host their exhibition. Predefined rooms are also provided, allowing users to skip the design process if they prefer.

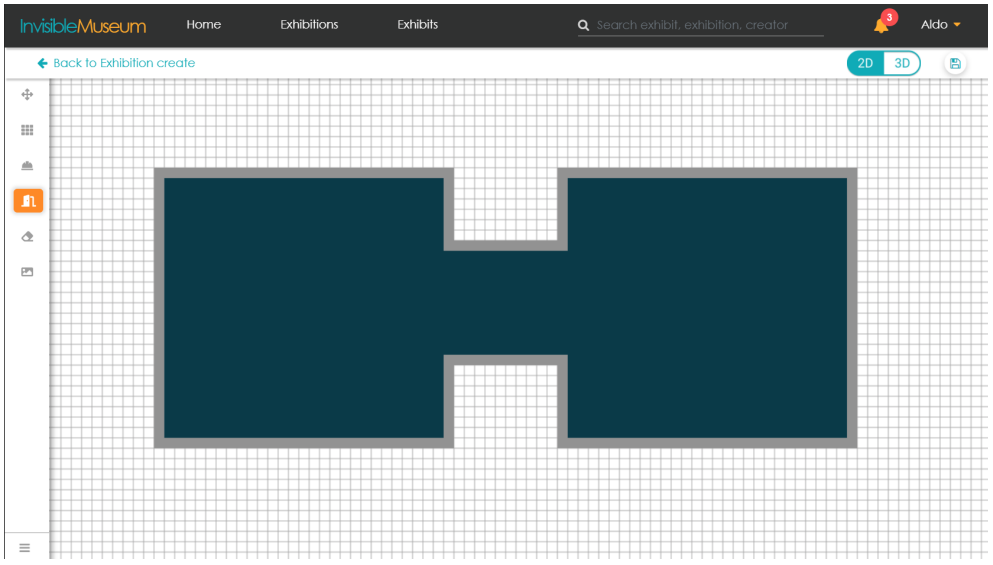

Figure 1. Floor plan of the exhibition space created using the space design tool

Adding Exhibits to the Exhibition

Once the space has been created, the exhibits to be included in the exhibition can be selected through the sidebar. Filters are also available, such as “My Exhibits,” “Favorites,” and exhibit type (3D, Images, Videos). Once the user finds the exhibit they want to add to the scene, they can drag and drop it into the environment.

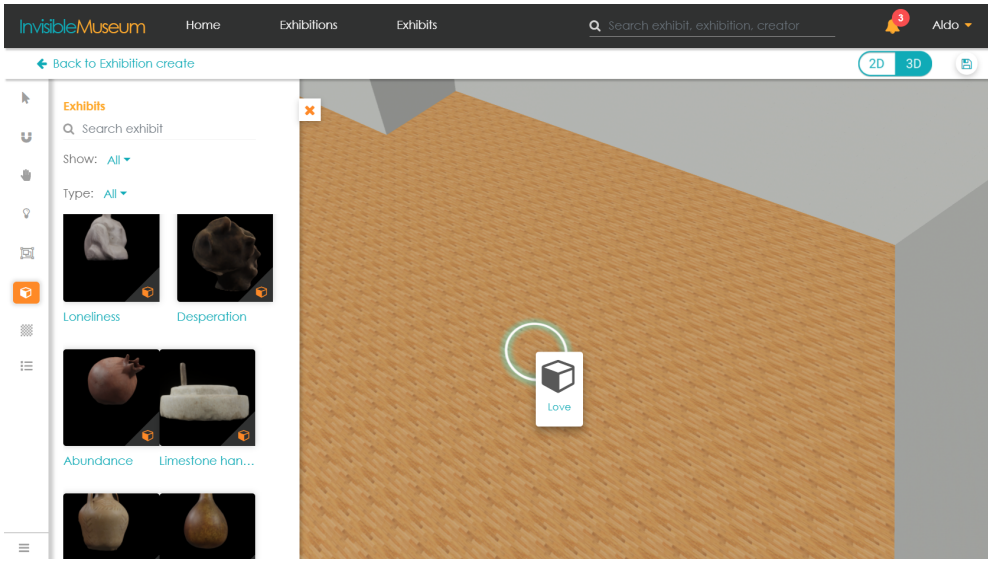

Figure 2. Illustration of the drag-and-drop mechanism for adding an exhibit

Editing Size, Rotation, and Position of Exhibits

When exhibits are added to the scene, they retain the size they had when originally uploaded to the platform. In most cases, these values, as well as their rotation and position, need to be adjusted depending on the requirements of the exhibition. Two tools provide this functionality: the inspector and the gizmos.

The inspector is a panel that appears whenever an exhibit is selected. It displays all relevant information about the selected exhibit's size, rotation, and position, while also allowing direct modification of these properties. It also includes options such as locking the aspect ratio during resizing and removing the exhibit from the exhibition.

Gizmos provide a more interactive editing experience. The user can click on an exhibit and select the appropriate gizmo mode through the sidebar or by using keyboard shortcuts. After selecting the appropriate gizmo for the property they want to modify, the exhibit can be adjusted directly using the mouse.

For optimal workflow, it is recommended to use gizmos for quickly setting up the scene, and then refine the adjustments with greater precision using the inspector.

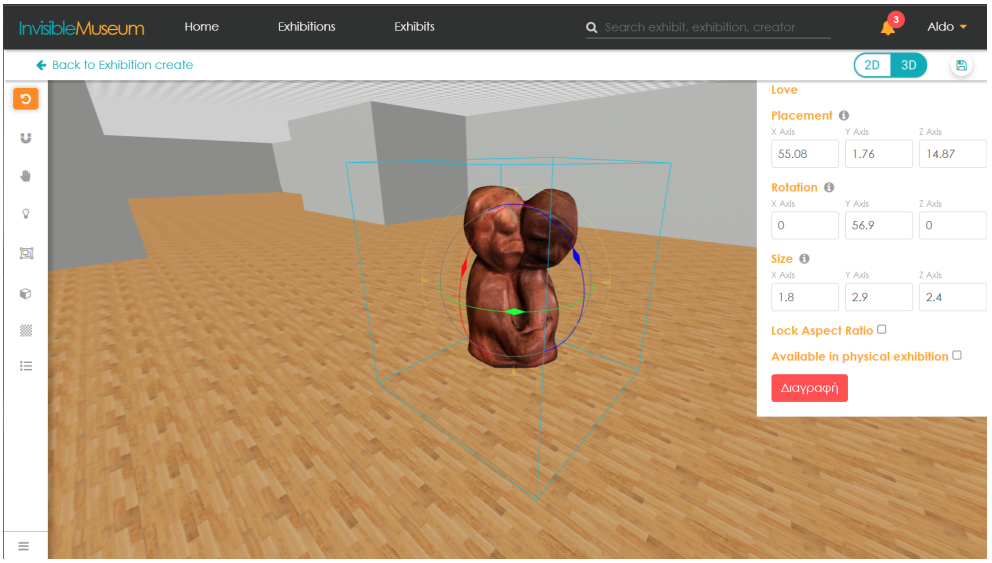

Figure 3. Selected exhibit with the rotation gizmo displayed on it. The inspector is also visible on the right side.

Adding Lighting

If the user were to add only exhibits to the scene, any visitor viewing the exhibition would see nothing but darkness. This happens because the editor includes its own lighting, which is not rendered for visitors. The issue is easily resolved by adding lighting directly to the scene.

Two types of lights are available. A general ceiling light that simulates a typical lamp, implemented as a point light, and a theatrical spotlight. Both types provide controls for adjusting the light's range and intensity. The spotlight also includes an additional parameter that allows adjustment of the beam angle, controlling how widely the light spreads.

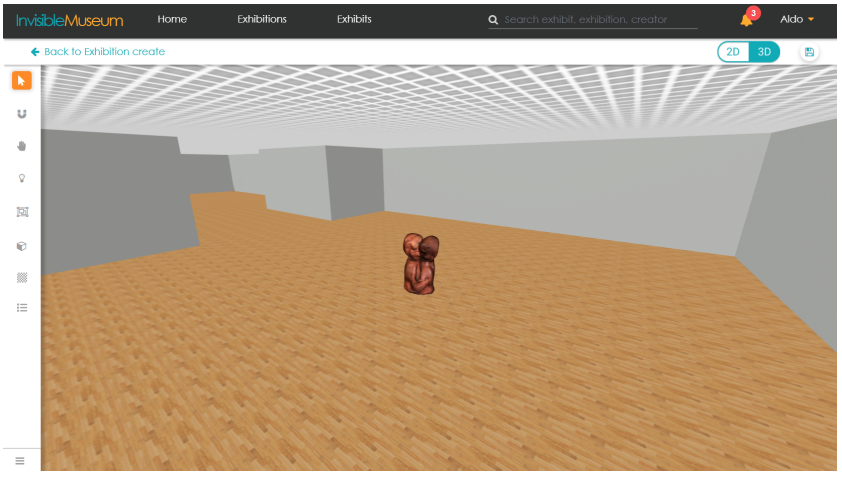

Figure 4. Exhibition without lighting.

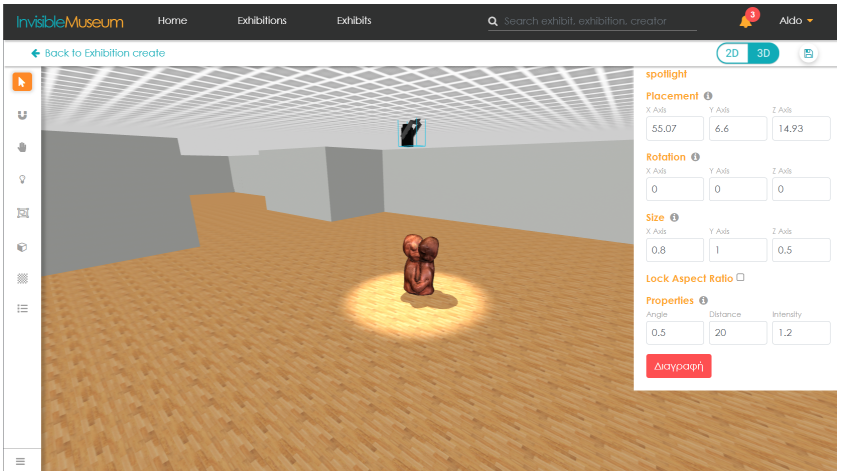

Figure 5. Exhibition with a spotlight.

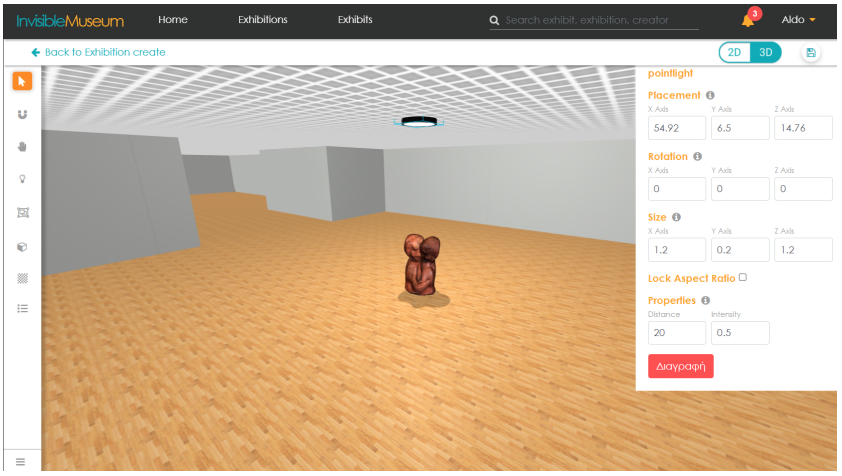

Figure 6. Exhibition with a ceiling light (point light).

Adding Decorative Elements

In order for exhibits to be added to the scene, users must first upload them through a separate page of the platform. This process requires providing metadata such as title, description, dimensions, and more. However, a user may want their exhibition to resemble a real-world space. This often involves adding models, such as a sofa, which are not actual exhibits. In such cases, using the standard upload process is not appropriate.

For this reason, the system provides the ability to add three-dimensional models, images, and videos during the exhibition creation process. This allows users to quickly decorate their exhibition either by uploading their own assets or by selecting from a list of pre-uploaded decorative elements.

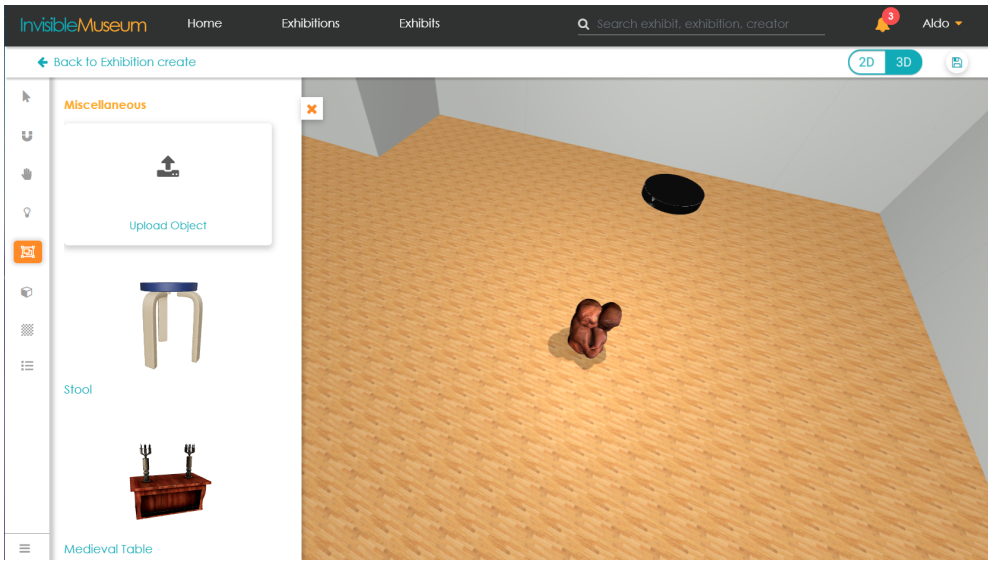

Figure 7. Decorative elements panel. The button for uploading user files is displayed at the top.

Camera Control During Editing

During editing, the user can control the camera in several ways. By holding the left mouse button, the camera can be moved parallel to the point of interaction. Using the right mouse button, the camera can be rotated around that point, while zooming is performed with the mouse wheel. These are the default controls, referred to as panning.

Through the sidebar or via keyboard shortcuts, the user can change the camera behavior by selecting alternative modes, including orbit control and zoom.

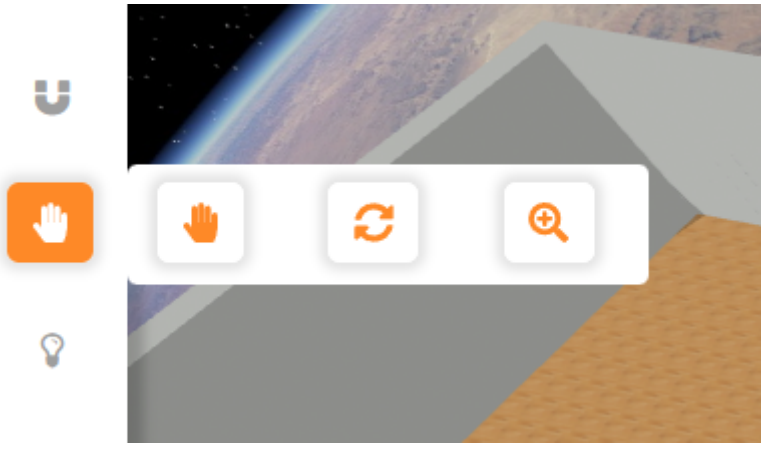

Figure 8. Camera control mode selection buttons.

Selecting the Tour Starting Point

After completing the creation of the exhibition, the user can create guided tours for it. The selection of the tour's name, description, and included exhibits can be done through a form. However, defining the starting point requires a visual representation of the exhibition.

For this reason, a dedicated interface was implemented, allowing the user to navigate through the exhibition using the keyboard. This enables them to select the exact point that visitors will first see when starting the specific tour.

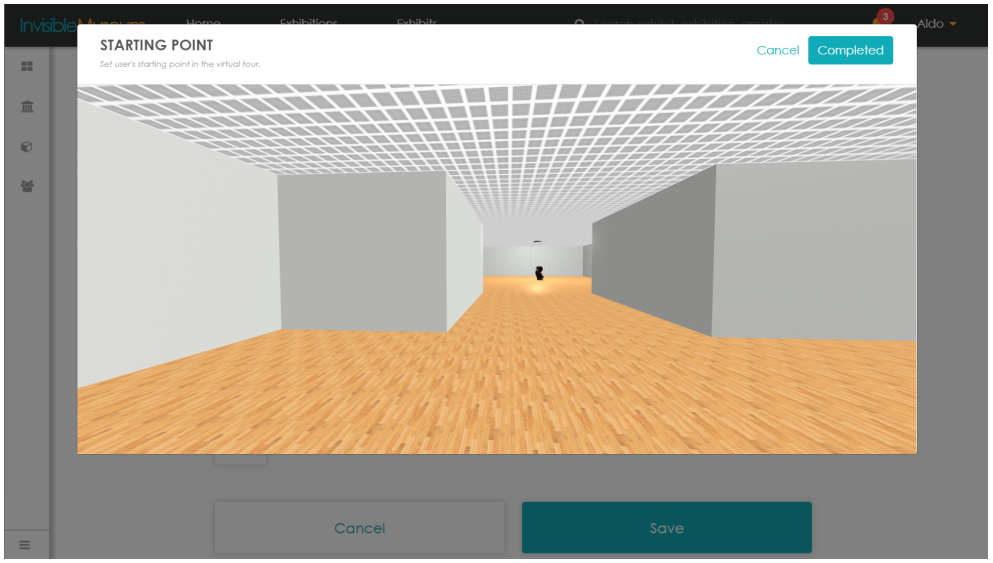

Figure 9. Starting point selection window.

System Implementation

Setting Up the Virtual Scene

Once the floor plan of the virtual exhibition space is completed, all the design information is stored in a two-dimensional array of integers. Each integer in the array corresponds to a specific structural element of the environment, as follows:

- 0 - Floor
- 1 - Wall
- 3 - Door
- 9 - Empty space

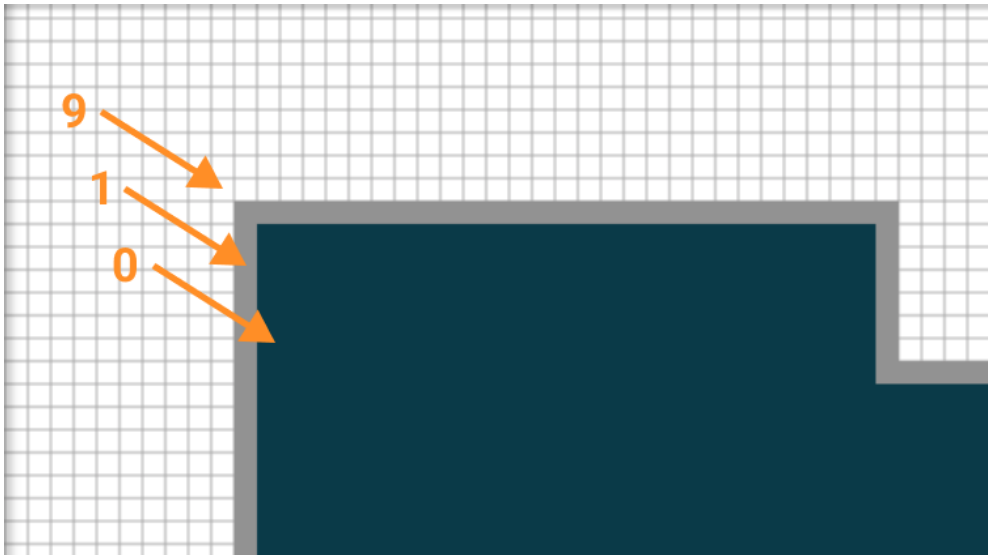

Figure 10. Mapping of canvas cells to numerical values based on their color.

Afterward, this array is used by an algorithm to generate the three-dimensional space. Each structural element is initially created as a simple box (cuboid). Once all components are generated, they are merged into a single object to improve performance.

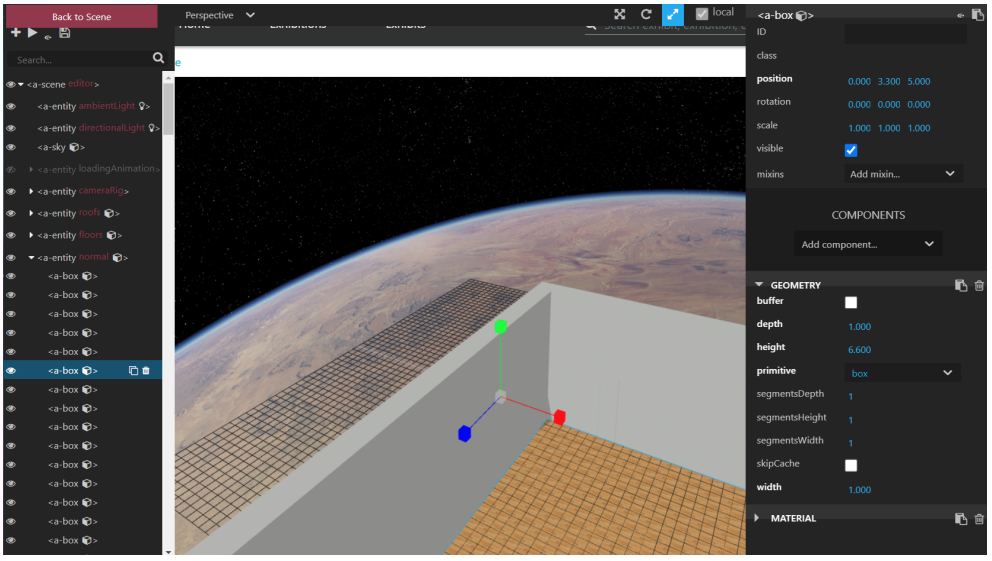

Figure 11. On the left side, multiple boxes can be observed being merged together to form a unified space.

At the same time, during this process, all exhibits associated with the exhibition are loaded from the database, if any exist.

Drag and Drop Mechanism

The implementation of the drag-and-drop mechanism relies on a combination of APIs. The first part of the implementation handles the transfer of information from the sidebar to the exhibition editor component. Angular provides a built-in mechanism for such interactions, but it is limited to communication within the same component. To enable data transfer between two different components, a service was created to facilitate this communication.

The second part involves indicating the position where the dragged object will be placed. This was implemented using the Raycasting technique, where the cursor acts as the source of the ray and the floor of the exhibition space serves as the target. At the point where the ray intersects with the floor, a visual indicator, in the form of a circle, appears. This helps the user accurately select the position where the object dragged from the sidebar will be placed.

Duplicate Exhibits

When an exhibit is selected from the sidebar and dropped into the scene, a request is sent to the database using the exhibit's ID in order to retrieve the associated model, image, or video file for display. This approach introduced a problem: if the original exhibit were deleted from the platform, it would also be removed from all exhibitions where it had been used, leading to inconsistencies and potential data loss.

To address this issue, a duplication mechanism was introduced. When an exhibit is dropped into the scene, a request is sent to the database to create a copy of that exhibit. The system then returns the ID of the duplicate, and the process continues in the same way as with the original item.

As a result, even if the original exhibit is deleted from the platform, the instance used within the exhibition remains intact and unaffected.

Camera Control System

For camera control, the Orbit Controls API provided by Three.js was used. Since mouse events are also utilized by the gizmos, it was necessary to distinguish between these interactions. This was achieved by enabling or disabling specific functionalities depending on the user's actions.

Orbit Controls are also closely integrated with the object selection mechanism. For example, when the user clicks on an object, the camera must recognize it in order to adjust its movement accordingly. Based on this, the "Focus object" feature was implemented. This feature allows the user to focus on a selected object by automatically orienting the camera toward it while working on it.

Finally, for improved usability, the cursor changes dynamically depending on the active functionality, providing visual feedback to the user.

Capturing Screenshots

Capturing screenshots is an essential feature for saving an exhibition. The screenshot is used as a cover image in the exhibition details form within the editor. It also serves as a visual indicator of the current state of the three-dimensional exhibition.

The implementation consists of two parts. The first part uses the API provided by A-Frame, which enables capturing a snapshot of the canvas where the scene is rendered. The second part involves converting the canvas into a binary large object (BLOB), allowing it to be included in a POST request and stored in the database.

To prevent the accumulation of multiple screenshot files in the database with each save operation, the existing image is replaced every time based on the exhibition's ID.

Loading Bar

The size of exhibitions can often exceed several tens of megabytes, leading to loading delays that are uncommon for typical web applications. To prevent users from assuming that something has gone wrong during loading, the implementation of a loading bar was necessary.

The basic calculation for the loading percentage is straightforward. It requires knowledge of the total size of the exhibition and the size of the files that have been downloaded so far. However, this approach introduces a limitation, since the size of downloaded files is only known after each file has fully loaded. As a result, when large files are involved, the progress updates occur infrequently.

To address this issue, an estimation based on the user's average network speed was introduced. With this approach, the loading bar updates dynamically according to the estimated download rate and the total size of the exhibition assets. Although this method does not provide perfectly accurate progress values, the discrepancy is minimal and not noticeable to the user, resulting in a smoother and more responsive loading experience.

Conversion from "Scale" to "Dimensions"

When exhibits are uploaded to the platform, three-dimensional models retain their original dimensions. Initially, the system was implemented in a way that allowed users to resize exhibits by applying a scale factor to the model's original dimensions. This approach caused confusion. For example, an exhibit with dimensions 10×10×10 could appear to have the same "size" as one with dimensions 3×3×3. In this case, "size" effectively represented the scale value, whereas it would be more intuitive if "size" corresponded to actual dimensions.

Transitioning from a scale-based system to a dimension-based one proved more complex than expected, since Three.js does not directly provide dimension data for models. To overcome this limitation, the bounding box functionality offered by the API was used. The bounding box enables the calculation of an object's dimensions after any scaling transformation.

The size is computed using the following relation:

Size = (Scale_final / Scale_initial) * Dimensions

Here, the dimensions are derived from the bounding box. With this approach, all exhibits are presented with consistent and meaningful size values, improving clarity and usability for the user.

User-Based Evaluation

As mentioned earlier, the system described in this thesis is part of a larger platform. During development, project collaborators used this system to test various components of the platform. Through this testing process, valuable feedback was generated, contributing both to bug fixing and to improving the overall user experience.

All available features of the virtual exhibition system were utilized during the creation of exhibitions. Initially, the two-dimensional space designer was used to construct the layout of the exhibition space. Then, the corresponding exhibits, previously uploaded to the platform, were added. Additionally, decorative elements such as display cases, murals, and lighting, including ambient lighting and spotlights, were incorporated. All elements were appropriately adjusted in order to fit the virtual environment and achieve proportions similar to those of a real-world exhibition.

Beyond internal testing and evaluation by the development team, an external evaluation was conducted at the Historical Museum of Crete. As part of a cultural program, the museum used the platform to create two exhibitions titled “Stelios Amanakis: Fighting on the Western Front” and “Odysseas Elytis: ‘Make sure it is printed identically’,” both based on real scanned exhibits.

During the creation of these exhibitions, several usability difficulties were observed, which led to scheduled on-site visits to the museum. The evaluation process included real-time system usage, during which users were asked to verbalize their actions following the thinking-aloud method. This approach revealed additional usage patterns that had not been anticipated during earlier behavioral analysis. Furthermore, users were asked targeted questions about any issues or difficulties they encountered, in order to systematically document them for future resolution.

The most significant issue identified was the disappearance of exhibits from the exhibition. This was caused by asynchronous saving to the database, which could overwrite a newer version with an older one. The reproduction of this issue required real users interacting with the system in ways not previously covered by existing use-case scenarios.

In addition, several smaller-scale issues were identified. While not critical, these negatively affected the user experience. These included visual inconsistencies, delays in loading certain functionalities, difficulties in using specific tools, and cases where users were unaware of the existence of certain features.

Conclusions / Future Work

This thesis was a significant challenge, but the final result is highly satisfactory and leaves ample room for further development and improvement. The main factors that increased the difficulty of the project included the lack of sufficient documentation, the complexity of combining multiple tools such as Angular, A-Frame, and Three.js, and, most importantly, limited prior experience with similar web-based projects.

On the other hand, the team working in parallel on the same platform provided valuable support, contributing to the smooth development process and the resolution of issues that arose along the way.

At its current stage, the project is functional and can be used by users. However, there are still features that could be added to further enhance the user experience. Key functionalities that have not yet been implemented include, for example, Undo/Redo mechanisms based on established design patterns, as well as collaborative editing involving multiple users. These features, along with others that may emerge, represent clear directions for the future evolution of the project.

References

https://angular.io

https://threejs.org

https://aframe.io