GAMIFYING ARCHAEOLOGY: An interactive 3D representation of Dura-Europos

Ali Hafez
ali.hafez@yale.edu

Advisors:
Holly Rushmeier
holly.rushmeier@yale.edu

Michael Shah
michael.shah@yale.edu

A Senior Thesis as a partial fulfillment of requirements for the Bachelor of Science in Computer Science

Department of Computer Science
Yale University
May 1, 2025

Acknowledgements

Special thanks to my family, my friends, and my professors, for teaching me everything I know and always pushing me to be the best person I can be. And thank you to Yale and the Department of Computer Science for cultivating an environment that facilitated my learning and has allowed me to realize my potential as a student, creative, and an academic.

Contents

Gamifying Archaeology: An Interactive 3D Representation of Dura-Europos

Ali Hafez

Abstract

In the 1920s and 30s, Yale and other university archaeologists excavated the site of the ancient metropolis called "Dura-Europos" in southeastern Syria. In the hundred years since, the scattering of artifacts, degradation and remoteness of the site, and instability in the region have necessitated a digital archive. Yale and Bard universities have been the primary patrons of this archive, with photographs and information recorded from the initial excavations, organized using linked open data. While there is a large amount of data available, the site is enormous and the sparse photography has taken a long time to organize so far. Due to the civil war in Syria, the site was heavily looted and destroyed, and access to and protection for the site has only recently resumed, and the Dura-Europos digital archive is the closest thing we have to the site in its original, recently unearthed state. Using the photography, documentation of artifacts, and other information gathered about the site, we create an interactive, 3D representation that can help users, from laypeople to academics, assemble the site spatially and better represent the site and its dataset. This representation is not only a procedurally-generated reconstruction of the site in 3D, but also allows users to display images and pictures taken from the site. Then, users can "paint" pictures taken of artifacts or from sites onto the walls of structures, and export these modified structures as 3D models. Eventually, this software will allow users to participate in refining and rebuilding the digital site, from placing artifacts in their buildings to making custom geometry for structures, creating a better virtual representation of Dura-Europos for all.

1 Introduction

Dura-Europos was a bustling Hellenistic, Parthenian, and Roman city in the ancient Fertile Crescent, established along the banks of the Euphrates. The metropolois existed from 300 BC to 250 CE, from the rule of Alexander the Great until late into the Roman empire. After being invaded and its population deported, the city was buried in the desert for 1700 years. Its ruins now lie on the banks of the Euphrates in southwest Syria. In the late 19th and up to the late 20th century, Yale and other archaeologists excavated the Dura-Europos site. They took photographs, created drawings and diagrams, and took artifacts from the unearthed site.

Since 2006, the Digital Dura Europos Archive – an initiative co-directed by my advisor, Holly Rushmeier – has been compiling the data from these archaeological expeditions. The goal of the digital archive is to label, categorize, and place each of the images and objects from these excavations. Archaeological investigations were a destructive process, plus the following 100 years of exposure to the elements. The artifacts are scattered throughout the world, and the site was extensively looted during the Syrian civil war. This all makes a compiled digital representation – where anyone interested can explore the site as a whole – incredibly important. Software could allow users to explore the city at-scale, make new links between objects, and correspond photographs to what they really depict.

There are existing digital representations of the Dura-Europos archive, created by other students, researchers, and academics. Given the size of the dataset and how much of the information lacks labels and connections to each other, new and improved visualizations are imperative to ensuring new information can be gathered and derived from the database. All continued archaeological investigation in the past few decades has been done digitally or using isolated artifacts taken from Dura-Europos, due to the war and degradation of the site.

This project will be the first to place these sites and artifacts in a 3D space that, at scale, recreates the entire site. This 3D site will be made using Godot Engine, creating information at runtime using data queried from the WikiData repository for the project. Various methods will be developed to display this data, including turning buildings into 3D scale structures, displaying images as textures, and placing all of this on a 3D recreation of the terrain. In addition, the software will have support for the Arabic language, the native language of the region, allowing people who live nearby or whose ancestors may have participated in these expeditions to use the software as well.

The result is an interactive 3D software where a user can explore the site, click on buildings, and get more detail about them, including their descriptions, associated objects and images, and have the ability to relate buildings, places, and images. The described software was created over the course of a few months, and the resulting application and its source code is available at https://github.com/earth418/dura-europos-3d.

2 Background

Dura-Europos has been a significant target of archaeological research and interest for over a century. The interest arises because of its uniquely well-preserved state, its diversity – the city contained a Christian church, a Jewish synagogue, and many other temples – and its multitude of artifacts. New technology seeks to allow archaeologists to mitigate the dispersal of these artifacts, namely the International (Digital) Dura-Europos Archive.

2.1 International (Digital) Dura-Europos Archive

IDEA is a team of researchers from various universities around the world dedicated to the task of creating technology to allow the continuation of work on this project. Media and data in the archive is stored in a variety of sources, like WikiData, Pleiades, JSTOR, and with the Yale University Art Gallery. And this data consists of photographs from the site, photos of artifacts, architectural drawings, location markers, text descriptions, and links between data, as part of a data scheme called Linked Open Data (LOD). The hope of the project is to – in their words – ”reassemble and re-contextualize archaeological information related to Dura-Europos.”[1] This mission is also exemplified by the project in this paper, which takes *reassemble* literally in the case of a 3D reconstruction of the site.

Pleiades contains a top-down map with location markers of ”points of interest” in the Dura-Europos site, and though this map isn't fully complete, it can portray the site and the level of detail necessary for its reassembly.

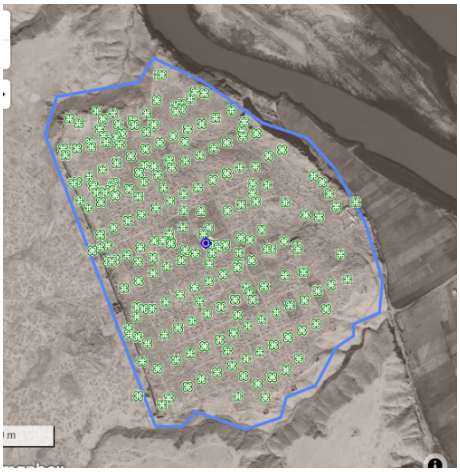

Figure 2.1: Pleiades list of locations in the Dura-Europos site

[1] https://duraeuroposarchive.org/

Additionally, the Yale University Art Gallery, which contains a large portion of the artifacts extracted and photographs taken from past archaeological expeditions, created a site that organizes some photographs and artifacts on a clickable map.

Figure 2.2: Yale Art Gallery's visualization of the history and artifacts of the site

2.2 Previous CS 490 Projects

In the past four years, several undergraduates have created CS projects to facilitate ongoing research on the site.

In Spring 2021, Patrycja Gorska made an API for future programming projects to query Dura-Europos data from ARTSTOR, since there was no existing API.[2] Zach Stanik conducted a project in parallel creating the frontend of this querying system, with a GUI to simplify these querying requests.[3]

This space has been intentionally left blank.

[2]https://zoo.cs.yale.edu/classes/cs490/20-21b/gorska.patrycja.pag52/.
[3]https://zoo.cs.yale.edu/classes/cs490/20-21b/stanik.zach.zcs8/

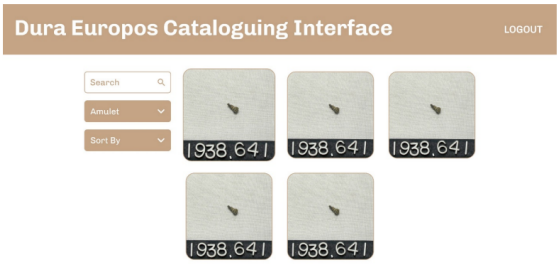

Figure 2.3: The GUI from Stanik and Gorska's project

That same semester, Qinying Sun created a GAN trained on stroke models created by a software called Photo-Sketching, in the hopes of automatically doing contour detection for stroke modeling across the site.[4]

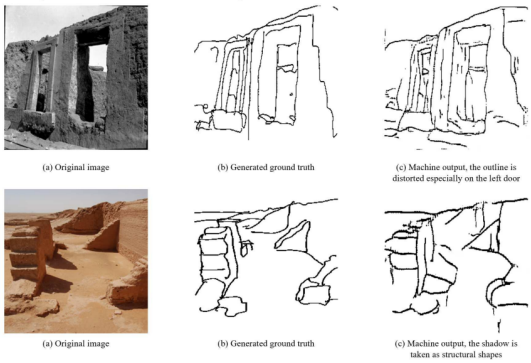

Figure 2.4: The results of Sun's contouring project

In Fall 2023, Mary Jiang conducted a project creating a dashboard that also including an annotator for images from the dataset, allowing users to add information and tags/descriptors to media and for that to be added to the wikidata repository.[5]

And finally, just recently in Fall 2024, Abagaial Tramer's CS thesis had a new organizational objective, displaying information from the WikiData site in a logical, understandable way, associating artifacts with sites and subjects visually.[6]

This space has been intentionally left blank.

[4]https://zoo.cs.yale.edu/classes/cs490/20-21b/sun.qinying.qs54/
[5]https://zoo.cs.yale.edu/classes/cs490/23-24a/jiang.mary.mj598/
[6]https://zoo.cs.yale.edu/classes/cs490/24-25a/tramer.abigail.at2285/

Figure 2.5: Mary-Jiang's dashboard and annotator

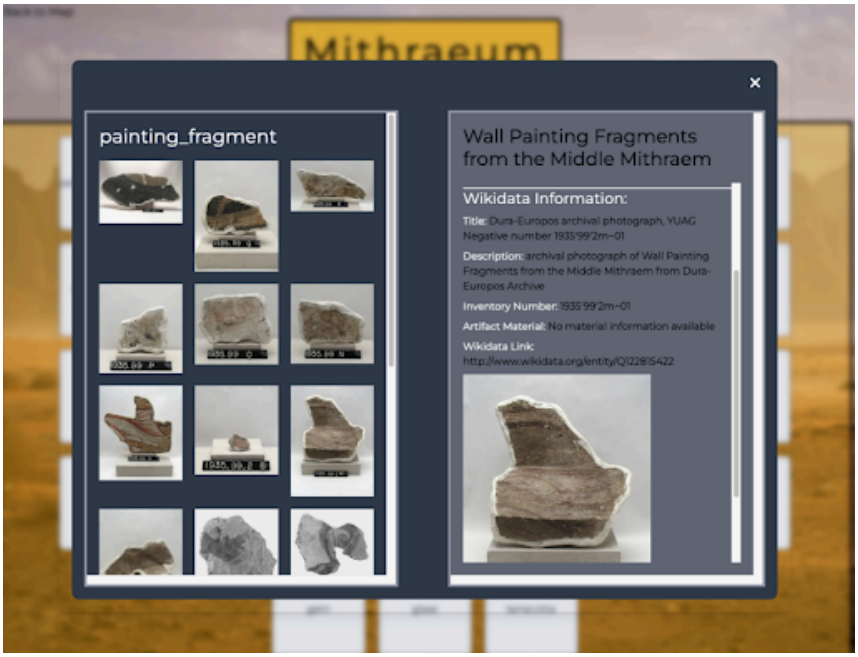

Figure 2.6: Tramer's CS 490 project, Fall 2024

3 Methodology

The project is largely an interpretation of the information in the Dura-Europos dataset, that takes various media and places them realistically in a 3D world, specifically in Godot Engine. The ultimate goal is a data-driven model that can use solely information from the dataset, regardless of what is added or removed, to completely display as many aspects of the site as possible in as much detail as necessary.

3.1 The World

The first step was to create a basis for the relevant area of the world. A Digital Elevation Model (DEM), a type of map that contains elevation/altitude data for each point in a region, was used. This data was obtained from the EU's open-access Copernicus Browser, as well as the added color texturing data.

A mesh was constructed from a heightmap extracted from the geographic area around Dura-Europos. I made sure the scale of this heightmap in-engine matched the real height scale of the terrain, to ensure robustness if different data sources are found, and to make sure the scale of the display was correct.

The Haversine formula was then used to translate from latitude to longitude. Information about objects, buildings, and layouts in the Dura-Europos dataset is stored in latitude/longitude and geoJSON – ensuring the translation to in-world coordinates was accurate, realistic, and correct, was imperative to the project, in line with the stated objective of robustness. The real radius of the earth and real elevations were used in the calculation and placement of these objects, so their scale in the visualization is the same as their scale in real life.

3.2 Structures

Once the base heightmap was created and textured with a satellite view of the area, the next objective was to overlay 3D instances of buildings, sites, and geoshapes. This was done by creating SQL requests to WikiData that obtained building information and geoshapes. These were cached and then used to construct the structures, each pair of points in the geoJSON becoming an individual wall (which was simply a cube scaled, transformed, and oriented to match the two points), eventually creating the entire structure.

3.3 Interaction

The final objective for my project was to create a method for users to interact with the re-created digital site, whether that be viewing additional information or visually making connections between spaces, objects, and pictures. Of course, this involved creating a UI to display information from the site, but I'll mostly be focused on 3D and interactive features.

The first step of this involved a mode where the user could "focus" on a certain building, with some information and the selected building's Q-number appearing on the side. Also, the user could revolve the camera around the building to inspect the walls of its exterior. In the future, the user would be able to switch between the fly-around and the revolving camera if they wanted to inspect buildings, and perhaps a "walking" camera would be added as well.

Also in this first step was creating the SPARQL requests to WikiData to enable the list of photos to be loaded, and their links and descriptions to be stored, so that when a picture is clicked, its ID (Q-number) is displayed, and its link is queried so that the picture is displayed.

The second step involved giving the user the ability to "pin" textures to walls – using photos from the site and clicked-on specified regions of them to the mesh walls, with a separate UI. This was done by getting clicked-on 3D locations on the building mesh, and mapping those in-order to a quadrilateral region of an image texture. Then, a plane would be created using those four clicked points, and its UVs mapped to the clicked UVs of the texture. This leaves something akin to a "wallpaper" of the texture on the wall, associating the pictures with an exact location on the mesh. And, because the 3D representation is exactly accurate in world space, an exact location can be gathered from these points, too.

The final step was enabling this resulting object – now with the textures painted onto it – to be exported to a 3D mesh. Luckily, godot all but has this feature built-in, which meant calling a function could export the building as a .glb file, which opens in Blender and verifiably has all textures supplanted onto it.

3.4 Other

Other, minute features were added as well. A user could request a custom photo from WikiData by typing its ID into the box, or search for a building using its ID and have it automatically selected. A compass was added for orientation purposes, as well. Many more quality of life features are expected to be added, and bug fixes to be incorporated, but that will require future testing and work.

4 Results

The project implemented many features to ease exploration and enable functionality in the application. These features were implemented according to the techniques described in the Methodology section. The world is constructed by geoshapes and a DEM model, displays Q-numbers and building blurbs, and enables exploration around the site.

The user can navigate around the constructed world by flying around freely. In the future, a mode to "walk around" may be added, particularly if building interiors are added.

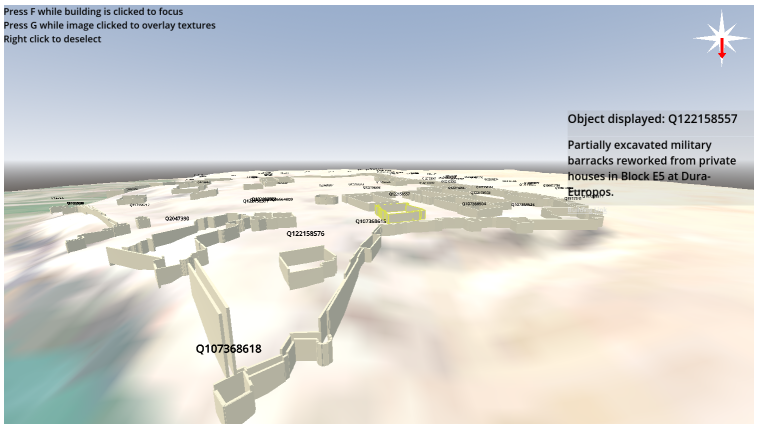

Figure 4.1: The world, with buildings containing extruded geoshapes and their Q-IDs.

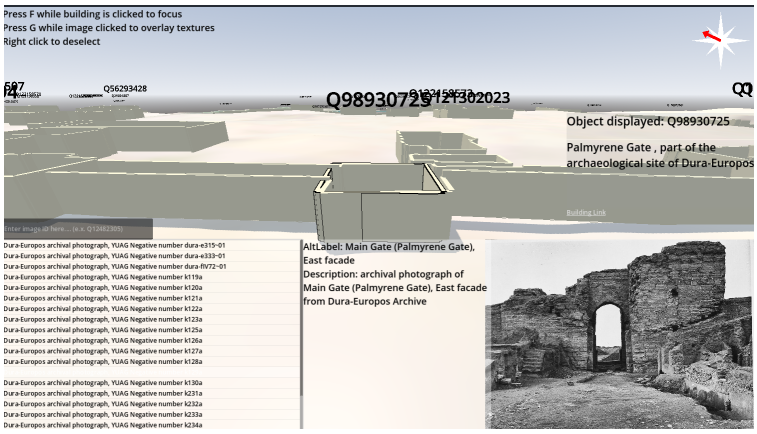

Figure 4.2: Palmyrene Gate and photo list/select display

When a loaded building is clicked on, the program will automatically request a list of all photos depicting it, and each one can be clicked on and seen, with their descriptions

displayed next to them. The camera will go into ”orbit mode” around the center of the building, to allow the ”focus” to be on the building. (Fig. 4.2)

The user, when pressing ”G” while a building is focused and a photo is open, opens up a separate menu, with a view of the picture on one half, and the 3D view of the focused building on the other half. The user can select any quadrilateral from the displayed photo, and pick four points on the focused building, and the photo will be placed on the mesh. Once this ”decal” is created, it can also be removed, or it can be exported along with the building as a .glb file.

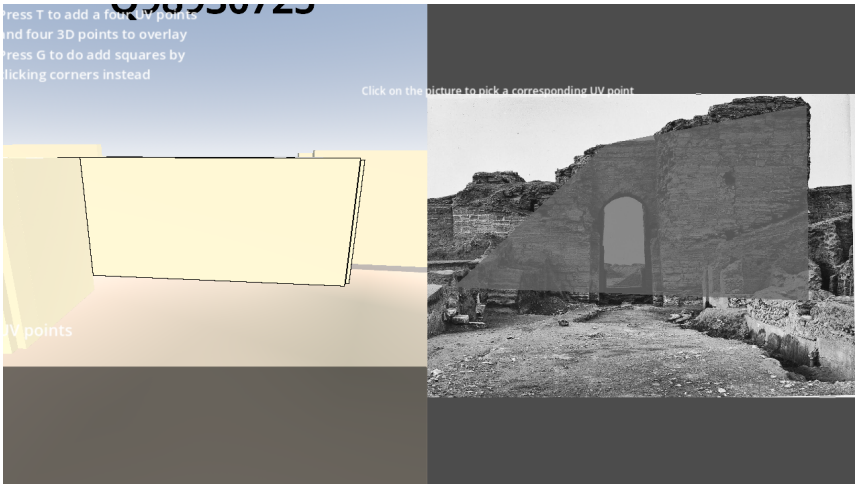

Figure 4.3: UV picking

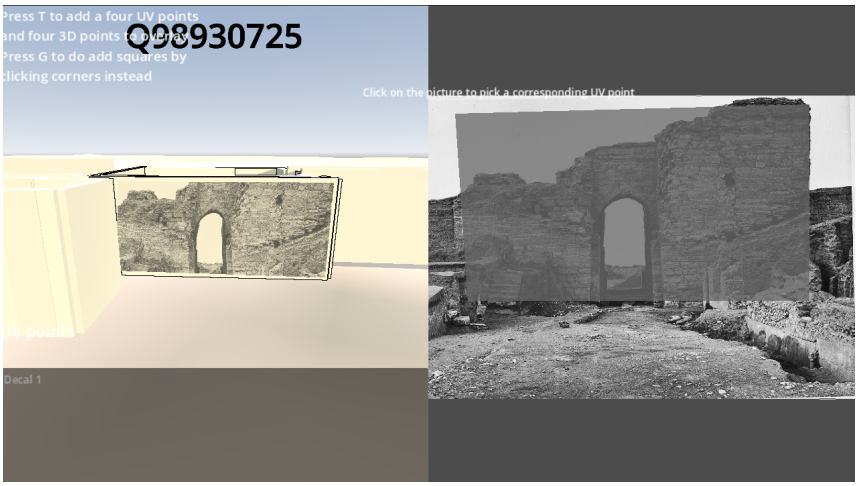

Figure 4.4: UV painting

4.1 Feedback

In late April, a researcher on the IDEA team was sent a minimum viable product of the application. She was easily able to get the application working, and while using it,

she compiled a list of bugs and desired features. These included issues with the camera and UI, some of which I was aware of, and some errors in functionality that I hadn't discovered. I fixed all of issues she brought to my attention to the best of my ability. She also suggested that I add a method to search for certain buildings, which I implemented, and a suggestion that I add a method in which a clicked picture could have a "suggested location" to be pinned to, which is something that was not added but that could likely be implemented down the line. The feedback was very important to developing plans for what will be done in the future on this project.

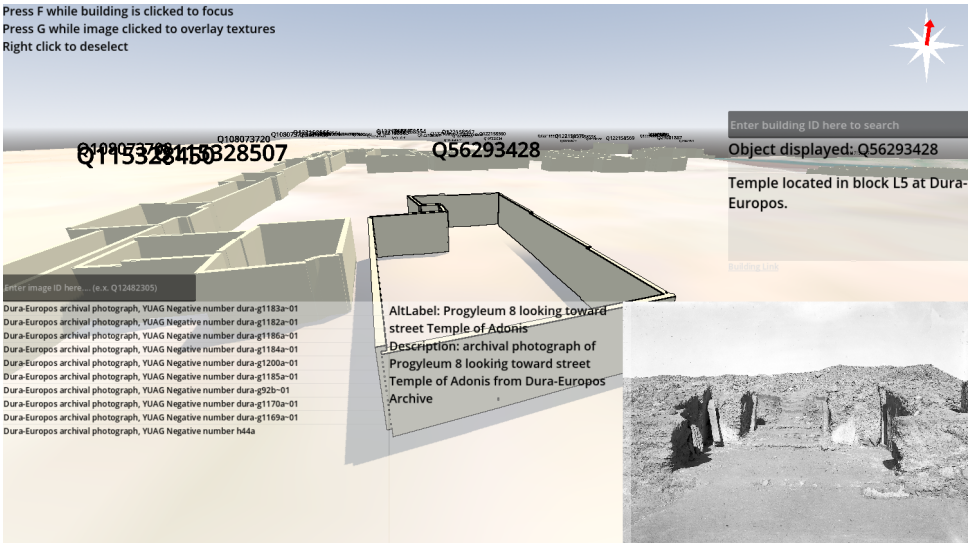

Figure 4.5: Final UI, with the search functionality.

5 Conclusions

The objective of this project was to improve upon existing methods for visualizing the Dura-Europos archive. This was done by creating a new, interactive, 3D software for users to explore the archive and a digital version of the site. Many of the project's goals were achieved, and the software is open-source, allowing future work to be done easily. According to the feedback of the researcher who tested the software, the project has the potential to have a lot of use, and the broader sentiment by the group has been one of excitement. The work continued rather smoothly without significant technical hiccups, and the resulting software is easy to use, with high potential for new features in the future. Eventually, gamifying archaeology will be a reality!

6 Future Work

Many features were conceptualized and not implemented in the interest of getting a minimum viable product to researchers in time for this project's completion. These included small customization features like interaction options, movement options, and camera options. And fixes to existing technology, like real-time requesting of buildings and displaying their geoJSONs (this has begun implementation, but currently the software can only use a cached set of buildings), as well as using the data to display the rooms inside of buildings, when that information is available. Additionally, it was intended that more bug fixing and quality of life fixes would be done, but as it stands, the application is usable, and its visualizations are useful, and its products can be exported to 3D files, confirmed by researcher feedback.

Feedback was received from a researcher on the project toward the end of April, and this feedback, while in the process of being implemented, gives a clear vision of what more future work would consist of. Lots of effort needs to be put into usability, user experience, and smoothness, to make the experience as good as possible for users, in particular non-technical ones. The camera in particular has a lot of usability features that could be improved. And some small suggestions were provided, in addition to the existing "wishlist" of features.

Of course, there are some obvious extensions of functionality that need to be worked on. There is currently no way to import a custom 3D model into the software. Textures can *only* be painted onto the structures and buildings, but some textures might be useful to place on the floor, and perhaps in more complicated or variable shapes. And there is no model for artifacts and their placement in the site, and in fact very little to do with interiors and interior detailing in general. Also, the city is a dense gridded city with many "blocks", not just the buildings that are there – in the future, even for buildings who might not have associated geoJSONs, a structure should be there. Overall, the realism and fine detail of the representation will be added by the WikiData and its users, but it's still important that the software facilitates this reconstruction process as smoothly as possible. Thankfully, this project will receive continued work both this summer and next summer, and I will be readily providing any advice necessary to the students working on the project. A rudimentary documentation for the codebase I created is also available for future students to reference in their use and further development.

A Code Documentation

I'm writing a temporary version of the documentation right here in the Appendix, just to make sure it exists somewhere. It's also in the README in the repository. Hopefully, this will be moved to a real documentation site later on. We'll go class-by-class.

A.1 GeoJSON_Mesh

In *geojson_mesh.gd*, extends Area3D. This class is intended to generate a 3D mesh/structure from a geojson file or text input, though using just text is still untested.

load_json_file(filepath) updates the **json_path** and loads the file's content into **json_contents** (as a JSON dictionary)

get_building_id() returns the id of the building, calculated from the json filename

refresh when this bool is clicked in the editor, re-runs the functions in the setter (load_json_file, generates_mesh)

clear bool clicked in editor that causes meshes to be cleared

generate_collisions (deprecated) whether or not a collision should be generated that is identical to the mesh structure. Deprecated because it should really just be on all the time – it has very little performance hit and is 100x better than the alternative (having it false).

(const) center_loc the center location of the map, as latitude longitude. Used with haversine formula to get 3D locations from geoJSON lat/lon pairs

lat_lon_to_cartesian(loc) uses the (real!!) radius of the earth, the center_loc, and the Haversine (distance) formula to get a Euclidian, real-scale location on the flat plane

generate_geojson_mesh() This function is kind of the meat of the class. It loops through each "feature" of the geoJSON, but for our purposes it should just have one – the building. First, it loops through the geojson coordinates, converting each lat-lon coordinate to a point on the terrain plane. Then, using the heightmap and bilinear interpolation, places their heights exactly at the altitude they would be at in real life – the map ranges from

174.5m to (174.5 + 57.3 = 231.3)meters in height, and so the heightmap result is interpolated to that range. The min/max was used for a simple bounding box collision, but that is not really used anymore. Then, the locations – now exactly corresponding to the map – are looped through, and for every pair of points, some math is done to find out exactly how to stretch a cube along that axis to make a "wall" between those two points. The height of the wall is arbitrary, but there is a minimum so that it doesn't look weird. The collision shape is a cube generated the same way, so it perfectly matches the cube that's representing that wall and can thus be clicked on with a ray trace. These are added to the scene using *add_child()* – the collisions must be added as a direct child to the Area3D, as far as I know, and the cubes are added to a *mesh_parent*, which is by default a child of the GeoJSON_Mesh.

A.2 DE_Meshes

in *de_geojson_meshes.gd*, just a Node3D. This class loops through the folder of geojsons and loads meshes from them. In the future, these files should be downloaded and cached automatically from the wikidata, using the below incomplete functions:

request_all_buildings, load_objects_wikidata, httpr, _buildings_request_completed, and _geojson_request_completed These functions are mostly implemented: request_all_buildings puts in a request that gets all buildings and the links to their geoJSONs. Load_objects_wikidata is sort of the master function that manages requests and calls. The other two functions handle HTTP requests from the wikidata site. The part that doesn't work is that every geojson_request returns a 303 error, even though if you click the links yourself, it'll take you to the building's geojson (if it exists). For a reference implementation that works, look at **picture_gallery.gd**.

object_list A dictionary that maps from building_ids (e.g. Q1324....) to a Node that can be edited.

load_objects Loops through every geojson in the assets/geojsons directory and creates a geoJSON mesh with it, using **create_object**.

create_object Create_object (and its companion **create_object_from_json**) create a geoJSON mesh with the specified JSON file/JSON content and add it to the scene tree, returning it from the function.

A.3 WikiGallery

In *picture_gallery.gd*, extends *Control* (meaning it's a UI class). Has to have at least children: **ItemList**, which is used as the picture list; a **TextureRect**, which displays the clicked picture; a **info_eg** which displays the picture description. Also planned is the

info_ar, for Arabic text, but unfortunately I did not fin da way to easily request two languages at once from the WikiData.

set_httpsignal() A helper function to simplify HTTP callback function management

sparql_request() Another helper function to simplify making a SPARQL query.

add_custom_image() Searches for an image typed into the "add-image" box in a SPARQL query. The HTTP calls **_image_id_request_completed** which adds it to the photo list if successful.

get_pics_depicting_building(building_id) Makes a SPARQL query returning all pictures tagged as depicting a specified building ID. **_building_request_completed** adds all these pictures to the photo list and stores their information and descriptions.

_on_item_clicked(index, ...) Callback function for the **ItemList** when an item is clicked. It gets the URL of the image from the associated array, sets the **info_eg** box to display the image's info (since that doesn't need any new requests), and makes an HTTP request: the callback **_image_request_completed** loads the image into the **TextureRect.**

A.4 Player

in *player.gd*, extends Node3D. Handles inputs, looking around, and interaction with UI.

get_pointed_object() Returns the object that the camera ray is pointing at

save_to_file(obj, path) Takes an object and uses godot's API to save the object to a file path. Used when the user uses CTRL+S to save a clicked object.

_input(event) Handles inputs and calls functions depending on each function. For the case of saving, also opens the file dialog to pick a location to save to.

focus/defocus_on_object(obj) Focuses on an object and switches between the free and orbiting camera, and updates the WikiGallery.

A.5 UVEditorWindow

in *uv_editor.gd*, extends *Window*. This is a separate UI window, split in half between a sub-viewport looking at the world and centered on an object, and the other half with a texture. This is used for UV painting. On _ready, the class sets up variables and loads any previous decals on the building.

uv_pos_list A complete dictionary of all UV ”paintings” in the session, organized by building ID.

polygon A 2D polygon rendered over the texture, displaying the subsection of the texture you're picking.

state (PickState: {Tex, Viewport, Done}) The class functions with a finite state machine, that is in the ”Tex” state until it gets all four clicked points on the texture, all four clicked points on the 3D model, and then it's done and displays the texture. The **working_uvs** and **working_pos** arrays are counters of how many tex points and model points you have clicked, respectively. The information about clicks is obtained by **_on_texture_input** (essentially _input but just for just the tex display/right half of the screen) and **_on_viewport _container_input** (same thing but just for the viewport/left half of the screen).

apply_texture() Uses the saved info in **working_uvs** and **working_pos** to create a plane using the four three points, set the UVs to the declared UVs, and creates a procedural mesh with those characteristics. It also creates a new material (and if necessary a new visualmesh) and copies the displayed texture onto its albedo texture, so that the image is displayed according to the specified UVs. It checks if the triangles are oriented correctly and otherwise flips them, then creates the mesh and saves it to the **uv_pos_list**.

list_item_clicked() and list_item_input() Manages the selection of previously created decals from the list, and allows you to delete them as well.

A.6 Other classes

heightmap_visualizer.gd This is a quick class I whipped up to, at runtime, turn the downloaded heightmap into a 3D mesh. This was just to make sure that any changes would be reflected in the software and both the structure placement and the visual mesh.

freelook_orbit_camera.gd I copied the code for both of these cameras and threw them into one class. I changed a couple of things here and there, but overall the functionality is the same. Their code is cited in the file.