UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

# CS/IT Honours Final Paper 2021

Title: 2D views from 3D laser-scanned buildings

Author: Sizwe Zwane

Project Abbreviation: SCAN3D

Supervisor(s): Patrick Marais

| **Category** | *Min* | *Max* | Chosen |
|---|---|---|---|
| Requirement Analysis and Design | *0* | *20* | 5 |
| Theoretical Analysis | *0* | *25* | 0 |
| Experiment Design and Execution | *0* | *20* | 5 |
| System Development and Implementation | *0* | *20* | 20 |
| Results, Findings and Conclusions | *10* | *20* | 15 |
| Aim Formulation and Background Work | *10* | *15* | 15 |
| Quality of Paper Writing and Presentation | *10* | | 10 |
| Quality of Deliverables | *10* | | 10 |
| Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*) | *0* | *10* | 0 |
| **Total marks** | | **80** | **80** |

# 2D views from 3D laser-scanned buildings

Sizwe Zwane
University of Cape Town
Cape Town, South Africa

## ABSTRACT

This project covers the conversion of three-dimensional3D mesh structure into various two-dimensional views. The two-dimensional view are used by various profession such as historians, architects and engineers. Through this, it will allow for the preservation of these heritage sites from the three-dimensional laser-scans which allow studying of the three-dimensional meshes in the future. In order to do this conversion, the Zamani project presented the building of a tool that allows for this. The tool produces the following views: section view, elevations and ground plans.

## CCS CONCEPTS

• **Views → Section Views, Elevations, Ground Planes**.

## KEYWORDS

Three-dimensional mesh models, Two-dimensional images, Multiple plane slicing, Clipping planes, Meshlab, Chapel of Nossa Senhora de Baluarte

## 1 INTRODUCTION

The building of structures is a field within engineering and architecture that has provided civilians with shelter for certain purposes for many generations. These shelters vary such as houses, mosques, pyramids, city halls and more. These structures are built from plans/blueprints usually drawn up by architects then engineers build the structure. In the past, there plans were drawn informally relative to the present or not at all, and they went directly into the building process. In the present, we require drawings or images of the structures build in the past for referencing or studying. This is where the Zamani Project using three-dimensional lasers-canning of the structure in to models that can be referenced.

Laser scanning is a technique used by the Zamani Project to model three-dimensional images of heritage sites around Africa and some in Eurasia. These heritage sites include Castle of Good Hope, Wonderwerk Cave, a Great Zimbabwe complex and many more. The Zamani Project uses these laser scanned images to preserve heritage sites because the sites over time deteriorate and, historians and other professions need to study/alter these structures.

Professions that are likely use these images are architects and engineers. They use various two-dimensional images of the structures such as section views, elevations and ground plans. Section views are vertical slices (cutting plane) of a structures to shows whats inside or hidden by removing or cutting away some part of a structure. Elevations are views of a structure from its sides. Ground plans are similar to sections views but using horizontal slices. The images are used to communicate the structure in a traditional architectural setting. In figure 1, it shows one of the most well known heritage sites in the world, the Parthenon in the Acropolis citadel, in it's various two-dimensional images and including perspective images. This is an image of the Parthenon in its prime or its restored state before all the deterioration. The contrast in it's current state can be seen in figure 2

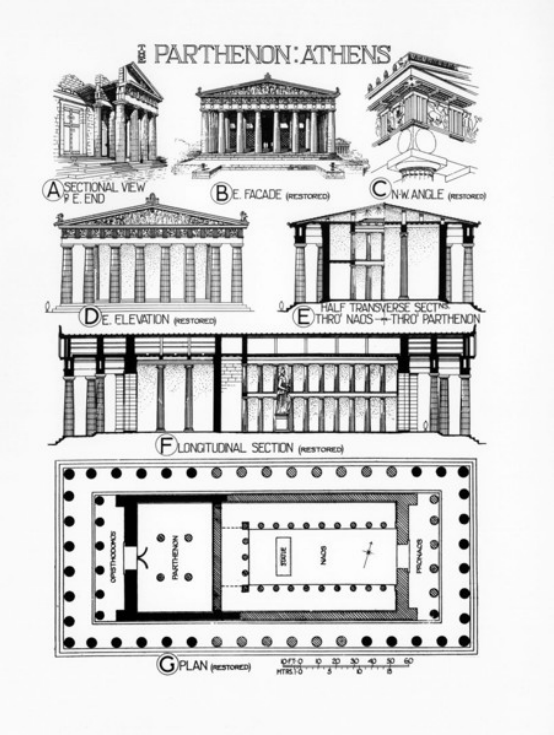



**Figure 1: The Parthenon of the Acropolis in Athens, Greece**

**Figure 2: Current state on the Parthenon**

## 1.1 Problem statement

Traditionally, this process of taking three-dimensional structures and producing these two-dimensional images has been done by architects in the context of what structure kind is being considered ,e.g. buildings, and this would take architects a couple of days in a lot of cases. However, has been no way of removing architects in the process of producing there two-dimensional images. The Zamani Project is looking for a tool of producing accurate and robust two-dimensional images form the three-dimensional data they collect from heritage sites. This tool will allow for the preservation of these sites and also studying them in the future.

## 1.2 Aims

The objectives of the project produce a tool/system that has a semi-autonomous way of producing these kind of two-dimensional images from very detailed meshes which have millions of faces which in a reasonable amount of time, the tool should be able to take three-dimensional models and produce its output as two-dimensional images and additionally it should include the produced images to have scale bar or grid, legend, date, a North arrow (is the information is provided) and a label/description.

## 1.3 Paper structure

Section 2 deals with the literature available on topics like mesh slicing and clipping planes foe the project. Section 3 outlines the requirements that are needed for the tool to have. Section 4 describes the implementation of the tool using various APIs/libraries in Python and other applications relevant in this tool. Section 5 explains the setup of the experiments conducted. Section 6 is the results and discussion part on the tool with various inputs. Conclusions drawn from the project are made in section 7.

# 2 BACKGROUND/RELATED WORK

The research amount behind producing two-dimensional images from three-dimensional structures is very limited in the field of Computer Science. The closest relevant research in the field has been used for layered manufacturing for three-dimension printing and, here the structures focused on are somewhat for small mundane objects such as figure 4 and has limited scalability with the algorithms ,i.e. the figure 4 mesh has approxiamately under a thousand faces where as the structures that we are working with have millions of faces.

## 2.1 Multiple slicing planes

In [2], Gregori et al. (2014) focuses on the efficiency of slicing triangular meshes with various algorithms. This is from the perspective of Additive Manufacturing, which is a process of building three-dimensional objects based on layering flat slices (or 2.5-D contours) of the mesh and his process is famously known as three-dimensional printing.

Since this paper focuses on efficiency, this is represented through complexities of the algorithms and a few parameters must be defined. $n$ and $k$ are the number on triangles and slicing planes respectively, also $\overline{n}$ and $\overline{k}$ are the average number of triangles and slicing planes respectively.

The simplest method of slicing is the trivial slicing by testing each mesh triangle along all the slice planes. The contour is recover by storing segments where there's triange-plane intersection for each slices and are sorted afterwards. The other algorithms compared are the Sweep Plane Slicing [3] and Triangle Grouping [10]. The proposed optimal algorithm is called the Incremental Slicing Algorithm. Intervals are formed by taking inclusive bounds of the highest and lowest $z$-coordinates of each triangle. An interval tree [11] is used to the intervals, and for a given plane value z, we use the stabbing problem [6] to retrieve all interval containing the plane. The tabulation of the algorithms in figure 3

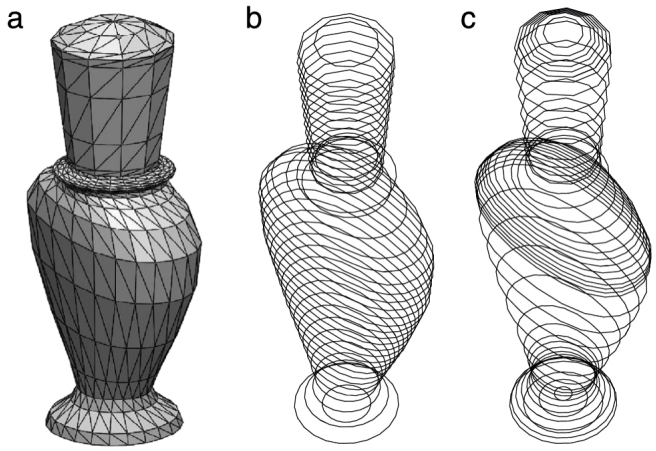



**Figure 4: The three-dimensional triangle mesh object (a) along uniform slicing (b) and adaptive slicing (c)**

## 2.2 Voxelization of mesh

Wang et al. (2006) [12] introduces a method slicing a voxelized model of three-dimensional mesh pattern. The mesh pattern made up of triangles. Each of the triangles is assigned a colour.

The method for slicing works as follows. It starts off by computing the contour polygon of the intersection of the triangular face by using three-dimensional polygon clipping to determine a necessary bounding box. Then determine the boundary voxels (figure 5) intersecting the faces and the boundary box. Then determine the contour polygon's area of the face's intersection and each boundary voxel's intersecting face. Lastly, for each triangular face, a colour must be assigned to each face then each of the voxels must be also assigned a colour based in the triangular face.

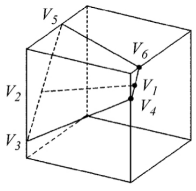



**Figure 5: Two faces intersecting on a voxel**

| Algorithm | Memory Construction | Worst case slicing | Contour assembly | Worst case total (assuming $k = \Omega(\log n)$) |
|---|---|---|---|---|
| Trivial | $\mathcal{O}(n)$ | $\mathcal{O}(nk)$ | $\mathcal{O}(\overline{n}\log\overline{n})$ | $\mathcal{O}(nk + \overline{n}\log\overline{n}k)$ |
| Triangle Grouping | $\mathcal{O}(n\log n)$ | $\mathcal{O}(n^2 + n\log n\overline{k})$ | $\mathcal{O}(\overline{n})$ | $\mathcal{O}(n^2 + n\log n\overline{k})$ |
| Sweep Plane | $\mathcal{O}(n\log n)$ | $\mathcal{O}(k^2 + n^2)$ | $\mathcal{O}(\overline{n})$ | $\mathcal{O}(n^2 + k^2)$ |
| Incremental Slicing (our method) | $\mathcal{O}(n\log n)$ | $\mathcal{O}(n\overline{k})$ | $\mathcal{O}(\overline{n})$ | $O(n\overline{k})$ |

**Figure 3**

Each of the triangular faces are described by three vertices, a colour and a normal vectors. The voxelization process (figure 6) become more computationally heavy as when the size of each voxel becomes smaller. Then processing voxels to enough accuracy is still a target.

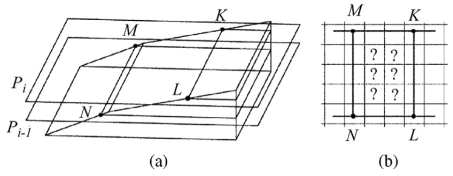



**Figure 6: Voxelization between two parallel planes**

The colourisation process of voxels helps simplify the identification of empty space versus the non-empty spaces. Identifying empty space is very critical for being able to project images that are further than the slicing plane. Voxelization consume large amounts of memory. Schwars and Seidel (2010) [7] are addressing the use of octrees for storage.

## 2.3 Clipping planes

In [9], Sutherland and Hodgman discuss producing two-dimensional images that are realistic to a viewer. There are multiple algorithms and approaches, but there are common steps. Such a perspective projections and polygon clipping.

To produce perspective images of object that are not transparent, information on depth is used to determine which parts of the object are near to the viewer. It is very convenient of producing three-dimensional perspective projection initially because this places the object in its correct position for the viewer while maintaining ordering via depth and also straight lines do not deform into a line that's not straight but they may change its length [8]. When applying perspective projections properly on three-dimensional objects, we treat them as if they are viewed from infinity. The formulation of a point $[XYZ1]$ into $[X'Y'Z'1]$ using perspective projection [5]:

$$[XYZ1]\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 1/d\\ 0 & 0 & 0 & 1\end{bmatrix} = [x'y'z'w']$$

$$X' = \frac{x'}{w'} = \frac{X}{(Z/d)+1}$$
$$Y' = \frac{y'}{w'} = \frac{Y}{(Z/d)+1}$$
$$Z' = \frac{z'}{w'} = \frac{Z}{(Z/d)+1}$$

where $X$, $Y$ and $Z$ are measured with the origin at the screen's centre and d is the distance for viewing, assuming conventionally that the user look in the towards the negative Z-axis.

Polygonal clipping lessens a polygon's surface beyond a three-dimensional boundary volume for viewing. The procedure removes parts of the polygon outside the boundary. Before clipping, perspective calculations only are not adequate produce the images. When a line connects a point behind the viewer to another point in front of the viewer, a strange line will be projected. A point on the same z-value as the view direction would project to infinite $X$, $Y$ and $Z$ which is not an easy point to represent. This will require limiting the allowed values of $x'$, $y'$, $z'$ and $w'$ before the division. Through these limits, then a range of possible values for $X'$, $Y'$ and $Z'$ can be made. The clip limit suggestions are:

$$-w' \leq x' \leq w'$$
$$-w' \leq y' \leq w'$$
$$0 \leq z' \leq w'$$

and the correspond to the six planes refer to as near, far, top, bottom, left and right and they form a frustum in perspective project using a set field of view that is half angle in $tan\alpha = 1/d$. Figure 7 shows the clipping planes.

# 3 DESIGN

## 3.1 Requirements

The requirements where gather by talking to our supervisor and kept referring to the project proposal. The requirements were not met easily at the beginning of the project. This also require using existing three-dimensional processing tool.

*3.1.1 Functional.* The following is a comprehensive list of functional requirements that the tool should include:

- Visualisation of the three-dimensional models to allow alterations
- Programming language to a develop the tool in with appropriate APIs/libraries
- Loading a three-dimensional mesh model

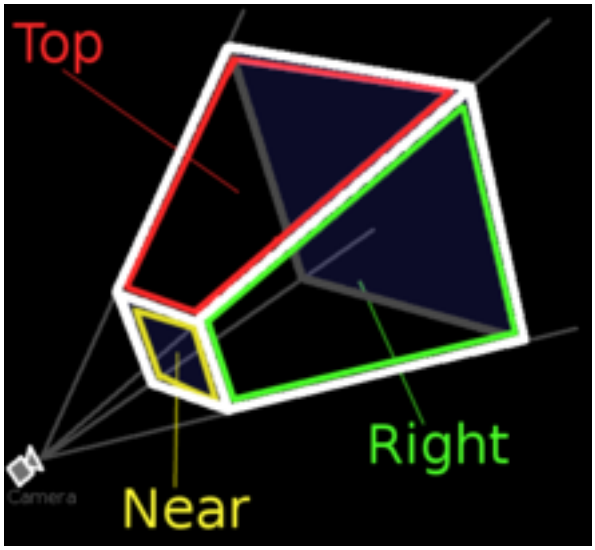



**Figure 7: Clipping planes**

- Using clipping planes to get the require two-dimensional images
- Visualisation of the two-dimensional images
- Correct field of view for the images
- Capturing the two-dimensional images
- Presentation of the images should include:
  - scale bar/grid spacing
  - date
  - legend
  - North Arrow (if provided)
  - short description of the two-dimensional image
- Conversion of the two-dimensional images into TIFF format

*3.1.2 Non-functional.* The following is a list additional requirements that the tool will include:

- Command line interface or graphical user interface
- Speedy tool that doesn't take unreasonable time to do its function
- Border around the two-dimensional images
- Visualisation of the three-dimensional models

# 4 IMPLEMENTATION

## 4.1 Overview

The project requires the use of some API/libraries in Python, namely Open3d and numpy . These can be installed using the package installer pip, but after installing the usage on the package is dependent on the Python version utilises for the project. Open3d has the following support only on Python, version 3.6 to 3.8. Also the open3d package has limited usage on some operating systems. The project also requires the use of Meshlab. Meshlab works as an easy visualization tool for users as its setup is heavily focused on the efficiency in processing through its interface. Currently, there seems to be not API/libraries in python that handle the procedure similarly to Meshlab.

## 4.2 Addressing the requirements

This part addressing the requirements individually by solving what required in the system through various classes, method, libraries or even other software.

*4.2.1 Three-dimensional visualisation.* There are a lot of three-dimensional visualization applications one can use. The most appropriate one for this is the Meshlab desktop application [1]. It is very simple to understand and this has an incredible user interface. The reason this requirements has overlap between both functional and non-functional is that is the model does require any alteration in the pre-processing stage then it can go directly to the main processing.

*4.2.2 Programming language.* The programming language used for this tool is Python because it has a library called *Open3d*[13]. This library helps with majority of the heavy lifting in this project is it will be discussed in the other requirements. Other relevant libraries are *matplotlib* and *numpy*[4]

*4.2.3 Load three-dimensional model.* This is done by the *read_triangle_mesh* method in *Open3D* from the *open3d.io* class. The method takes is a string containing the filename of the three-dimensional model and returns a *open3D.geometry.TriangleMesh* class type instance containing the vertices and triangles of that mesh model.

*4.2.4 Clipping planes.* This is done in the *open3d.visualization.SelectionPolygonVolume* class. The properties of an instance of this class need to be set. For this project, four boundary points are needed to cover the requires volume. These four point in an array are required to be converted into "float64" types. Then the properties *axis_max* and *axis_min* need to be set using the minimum and maximum of a selected component of the vertices respectively. The *boundary_polygon* property set using the *open3d.utility.Vector3dVector* class instance supplied with the "float64" type list. And finally using the *crop_triangle_mesh* method, it will return the clipped mesh after supplying the loaded model.

*4.2.5 Two-dimensional visualisation.* Open3D has a class *open3d.visualization.Visualizer* that deals with this requirement. One has to create a window using the the $create_window$ method, and supply the method *add_geometry* with *open3d.geometry.Geometry* type object which includes *open3D.geometry.TriangleMesh* types. The window must be destroy of using the *destroy_window* method

*4.2.6 Field of view.* For the project, it requires orthographic projections of the three-dimensional model but the default in *Open3D* is perspective projections. So, this needs the field of view to be change. First, the controller of the visualizer must be retrieved using the *get_view_control* method then change the field of view using the $change\_field\_of\_view(step = -90.0)$ method.

*4.2.7 Capturing two-dimensional images.* This done while rendering is happening in the scene. Use the *capture_screen_image* method by supplying a image filename. This method can only handle JPG and PNG image file types.

*4.2.8 Scale bar.* This is done using the *matplotlib* library. It has a *ScaleBar* class supplied with required parameters to the plot. The

two-dimensional image has to be loaded first using the *imread* method.

*4.2.9 Date.* This is done using the *datetime* python library. The current date is used as a string. The string is passed as text to plot on the top left corner.

*4.2.10 Image Description.* This is a simple use of the title method of a plot.

*4.2.11 Convert to TIFF.* After plotting the two-dimesional image using *matplotlib*, image is saved as a TIFF image format with reasonable quality.

*4.2.12 Usability.* The tool currents has a command line interface by by supplying the three-dimensional model filename.

*4.2.13 Border.* This is done using the axis of the plot in *matplotlib* but the numbering on the axis is removed and the axis is stretch appropriately.

*4.2.14 Performance.* The tool is respond fast with an initial two-dimensional image from the visualiser.

*4.2.15 Compatibility.* The tool uses only the required version(s) of Python a that allow for the use of *Open3D*, *numpy* and *matplotlib*

## 4.3 Additional implementation(s)

*4.3.1 Zoom.* The zoom constancy of the tool is built around the Chapel model using the *set_zoom* method from the *open3d.visualization.ViewControl* class. If a two-dimensional image is too wide for the window view, then track pad input of two fingers towards and away each other can be used to alter the image.

## 4.4 Pre-processing

*4.4.1 Step 1.* The three-dimensional model in a PLY format needs to be loaded into Meshlab. Assuming the image has geo-referencing, the top view of the image should be facing the user and do not rotate using the a click and hold input from the cursor. If this happens this step has to be repeated

*4.4.2 Step 2.* The model needs to be oriented such that the external side are parallel/perpendicular to the horizontal or vertical axis of the screen. This is done by rotation the model on its z-axis about its centre. To do this one has to go to the toolbar Filters -> Normals, Curvatures and Orientation -> Transform: Rotate. Change "Rotation on" and "Centre of rotation" to "Z-axis" and "barycenter" respectively (the barycenter ensures the model is rotated around its center because some models are not at the origin). The enter the angle of rotation, e.g. with the Chapel model, an approximation was made relative to one of the walls, it needed about a 30 degrees clockwise towards the horizontal axis and eventually settled on 34.5 degrees using a protractor (so -34.5 was entered ). The preview box was ticked to if there are any further adjustments needed. The colour of the model seemed to be gone afterwards but this is not important as long as the required side is oriented correctly. Now, export the model with the rotation using "Export as" in the "File" toolbar. This is precautionary to further unnecessary operations on the model.

*4.4.3 Step 3.* The model must loaded post rotation. Now what's required is to remove the obstacles that don't form part of the main structure e.g. The Chapel model is located on the coast of Mozambique and there are boundaries/fences towards the ocean that are included in the model ( figure 9), these boundaries need to be removed. To to this one must select the unwanted faces and remove them. The buttons for selecting and deleting unwanted part of the mesh are on the top right corner of the Meshlab interface shown in figure 8

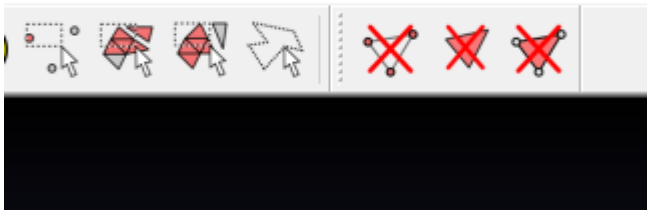


**Figure 8: Buttons for selecting and deleting**

**Figure 9: Chapel of Nossa Senhora de Baluarte in Stone Town, Island of Mozambique**

Then the model needs to exported similar to step 2.

*4.4.4 Before and after.* The figures 10 and 11 show what to expect when the pre-processing is done. The boundaries/fencing has been remove and a new orientation has been applied. Some models might not require extensive amounts of pre-processing but what's important is the orientation of the model.

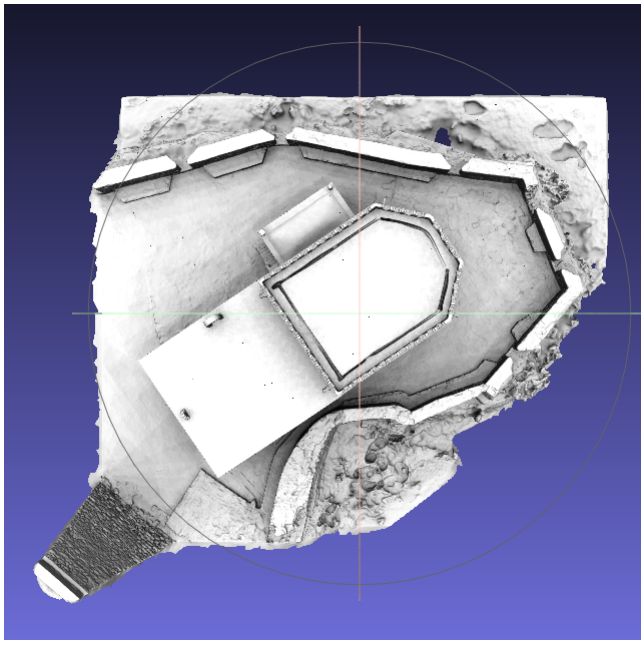


**Figure 10: Before pre-processing**

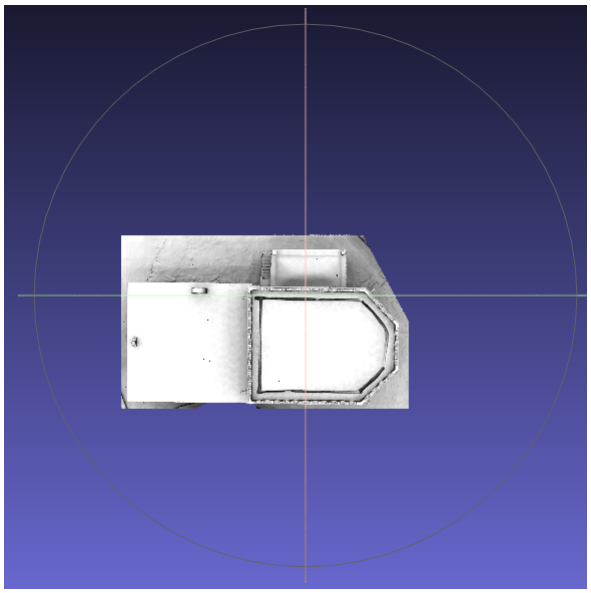


**Figure 11: After pre-processing**

## 4.5 Main processing

This section has been detailed in the figure 20 with an Architecture Diagram.

*4.5.1 Elevations.* The method to obtain the elevations by using the already oriented model with top view facing the user. The model is bounded by clipping planes on it's max and min coordinates. The clipping planes are set by selecting two z-values the model must exist in, in this case they just happen to be the max and min z values. The first image obtained is the top view, then the model is rotated along the x-axis by -90 degrees to obtain one of elevations. This elevation should be the one from down when looking at the top view. After this elevation the model is rotated along the y-axis (90 degrees) three times to obtain the other elevations. The bottom view of the model is not obtained. The images of the five elevations are saved for later processing. In the context of the Chapel model, figure 13 displays one of images to expect when a window pop ups.

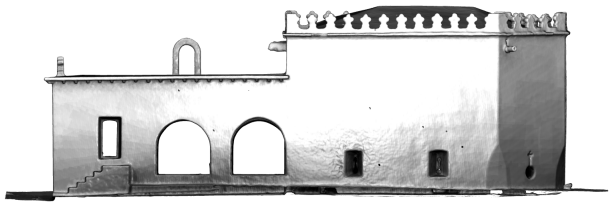


**Figure 12: One of the elevations of the Chapel**

*4.5.2 Section views.* The already oriented model is rotated about the x-axis like the first elevation.

The first section view is obtained by placing the model in a clipping plane that below its max z-value e.g. the near clipping plane is moved by shifting its z-value negatively unlike the elevations.

The second section view is obtained by rotating the model about the x-axis and y-axis by 90 degrees then clipping the models.

Both section view cutting planes are orthogonal to each other. The two view images are saved for later processing.

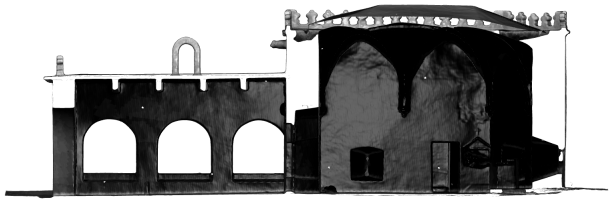


**Figure 13: One of the section views of the Chapel**

*4.5.3 Ground plan.* To obtain an image of a ground plan, the model is place in the placed in the clipping planes where the near z plane is moved around to find the optimum place of the cutting plane. If the building has multiple storey this process has to be done with multiple near z-plane movements.

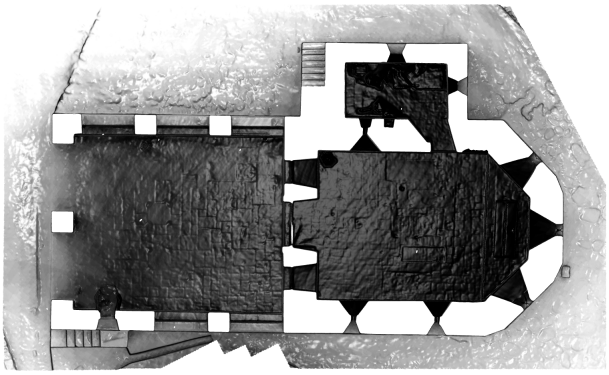


**Figure 14: Ground plan of the Chapel**

*4.5.4 Post-processing.* This part is more of the presentation of the images. The images requires a scale bar, date and label/title for each of them. The scale bar is uniform as each view is its own image. The images also require a border. On the Ground plan the elevation lines and the section view lines must be included. Finally, the images are converted into TIFF Format

## 5 EXPERIMENTAL DESIGN

The setup was running the tool on different three-dimensional models. The main models were the Chapel of Nossa Senhora de Baluarte in Stone town, Mozambique; Green Vault Museum in Dresden, Germany; and central Paris Police Station. This part has two stages.

### 5.1 First Stage

Since the models require pre-processing first. They will need to be unwanted parts before the use of them in the main processing. This is done using Meshlab and for each of the model this provided the data in table 1, which is numeric information and the make up of the mesh models. The result after this stage are in the "Meshlab Snapshots" row in table 2

### 5.2 Second stage

This stage is entered if the mesh models have been removed of the unnecessary parts to them. The processing in this stage is done by the tool designed and with the mesh models it produced some of the two-dimensional images in the "Elevations" and "No Vertex normal" rows of table 2 by varying the lighting in the scene. The run times for each view type is shown in figure 15 and there percentages in figure 16. In figure 15, the reason behind the hike in run time for elevations is due to the tool producing five two-dimensional images where as for ground plans it produces one and two for section views.

| Building name | Laser-scanned | No. of faces | No. of vertices |
|---|---|---|---|
| Chapel | Yes | 4,940,057 | 4,702,858 |
| Green Vault | Yes | 392,556 | 195,674 |
| Paris Police station | No | 6,367 | 4,793 |

**Table 1: Building Information**

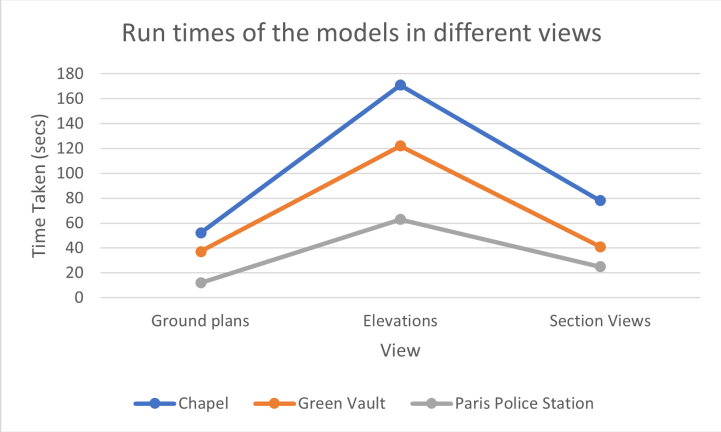



**Figure 15: Models' run times**

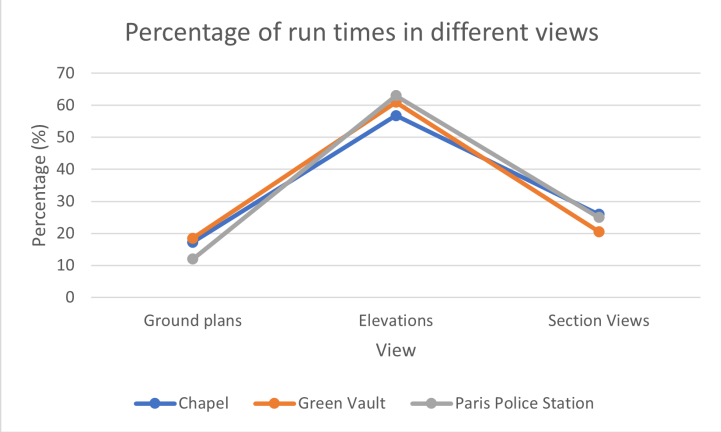



**Figure 16: Percentage of run times**

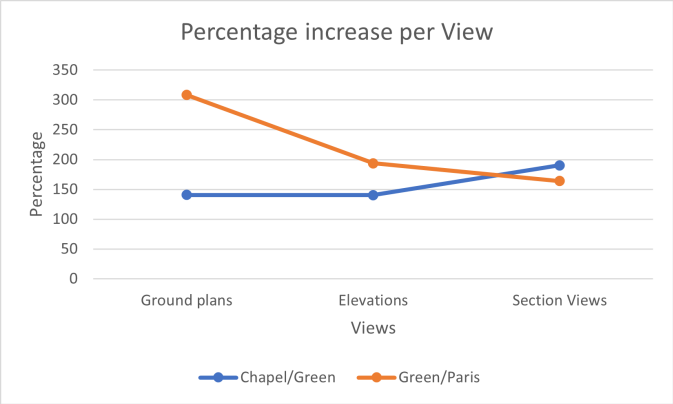



**Figure 17: Percentage increase**

In the appendix, page 15 and on wards show the results the tool produce with all the other fine presentation details from the Chapel model

## 6 RESULTS AND DISCUSSION

This section makes comparisons and deductions from the experiments using both visual and numeric data from the three-dimensional models.

### 6.1 Scaling

The scalability of the tool shows how much it can handle in terms of the input mesh models. The tool has good scalability because from the figures 16 and 17, it is able to handle consistent run time percentages in each view and the increase in the mesh size did not allow for an increase that greater than linear time, i.e. the Chapel model has about twelve time the number of faces and twenty four times the number of vertices relative to the Green Vault model, and had an increase of about 145%, 145% and 200% in the run times respective to the ground plans, elevations and section views.

### 6.2 Lower quality models

This tool heavily dependent on the manner in which a three-dimensional model is prepared. It needs a lot of details from the model to produce a great image. Features such as textures and colours on the models make for the produces image to have more detail. Models that tend to not have these feature not produce the desired images. In table 2, there are three buildings in their various image types and in table 1 there's some numeric information about the models. The Chapel and the Green vault are laser-scanned with varying quality but the Paris police station is not. Laser-scanners can take up to a million points per second so this indicates that there was a massive reduction in the model quality with the Green Vault post the scanning. Reducing model to this level is not great for the user's of the tool as this makes the model seem as if it was made by a human from sculpturing clay. This will not be a great way to communicate these types of structures and works towards the detriment of the preservation of these heritage sites. The professions that might require the usage of these models might not use the services if the model lose great quality. In future, models that lost much of there quality might need to do new laser-scans on the buildings if they are still intact in some cases. Now, with the Paris police station model, it wasn't laser-scanned and it's likely that it was built from scratch using software such as Maya or Blender. This can be deduce also by the number of vertices and faces the model has. The model has texture images that the tool seems to not capture and resulting in the elevation of the model having no fine details such as the windows and doors. This is a weakness of the tool as it is not able to handle texturing using two-dimensional images.

### 6.3 Efficiency

With tools like these, speed is a very important factor. It is assumed that the main reason a project like this is proposed is to find a faster way of achieving the desired results. The results being able to produce two-dimensional images. Again, traditionally one would need and architect or engineer to produce theses two-dimensional images. The process for making these two-dimensional images would take hours even days in some cases. This is where this tool triumphs over traditional means of producing these images. When using the Chapel model as input the tool was able produce these images in well under ten minutes consistantly. It took about three hundred seconds (five minute) in the iteration of figure 15. With images such as the floor plans and the section view, having architect produce such images would require them having to measure physically, and then draw the images for hours afterwards. Drawing these images would require an understanding of how slicing planes work on three-dimensional images and implementing them. Slicing planes take a while when having to drawing one from scratch. Even the multiple slicing planes algorithms referenced in the Background took when running them on a much simpler, less detailed model of the Colosseum in figure 18 and 19 . This model has about 250,000 vertices and it took about 20 minutes to produce a single slice (figure 19) . In contrast, using clipping planes is almost immediate.

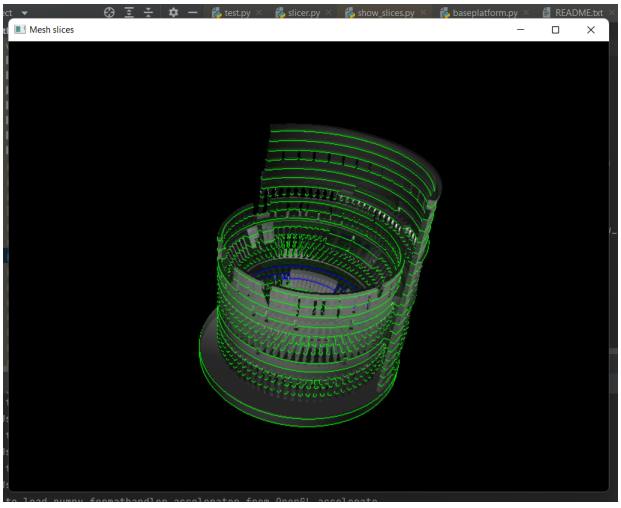



**Figure 18: Multiple slices (green and blue) around the Colosseum model using uniform slicing**

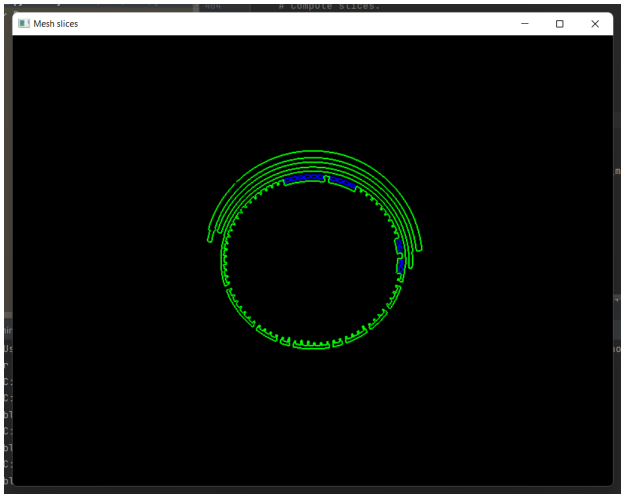



**Figure 19: Isolated slice of the Colosseum model**

## 6.4 Scene lighting

The lighting of the scene is very crucial to how the two-dimensional images are showed. In this case, various lighting setting were tested in the scenes to see which ones are better for producing the best quality possible. In Open3d, working with the lighting one has to adjust the settings using the *compute_vertex_normal* and *compute_triangle_normal* methods in the *open3d.geometry.TriangleMesh* class. The informal default of lighting the seems to be the *compute_triangle_normal* method, and if one explicitly calls on this method there would be no difference from the the default. In table 2, the images in the "no vertex normals" row represents the various buildings with the informal default.

Now, using the *compute_vertex_normal* method, the class computes the vertex normals for each vertex in the three-dimensional triangle mesh. The images in the "Elevations" row represent the what happens when this method is called. There is a massive difference in the appear when this method is called. The scenes seem to displays much greater/better detail in the images relative to when this method is not called. The images are less dependent on colours of the three-dimensional model, and this is apparent from the images with uniform colouring (the Green Vault and Paris police station) only display the images with only the outline of the building show where as with the Chapel there's various colouring on the object is still very detail but some detail is lost, e.g. the roof of Chapel is not displayed.

With the perspectives considered, an advantage of images made by architects is that when they draw the such images the emphasis is on the outline/borders are on a per face basis, i.e. if an architect draw's a triangle, they would draw the outline bold then colour the inside of it, where as when a scene is rendered, the triangle is one colour without a bold outline.

# 7 CONCLUSIONS

The research objectives of the project were to create a tool that can convert three-dimensional meshes into two-dimensional images of heritage sites using PLY format into TIFF format and additionally should have thinks such as scale bar, date and description. The objectives of the tool were met using the various programming languages, APIs/libraries and applications. But depending on the kind of data supplied to the tool, the quality of the two-dimensional images varies. The tool will allow for the Zamani Project to produce views such as section views, elevations and ground plans in great speed. So a lot more models can be converted into two-dimensional images with good accuracy.

The way in which the input mesh model is prepared is very important. The determines the amount of detail a produced image has. The tool has the best response to the data supplied by the Zamani project, the Chapel model. The result from it were incredible compares to the other models as seen in table 2.

The quality of the produced images is also dependent on a hardware component, the screen resolution. When the window pops up to capture in image, there are only a limited number of pixels a desktop/laptop can capture. Higher resolution screen can assist in capturing the finer details of an image as they will have more pixels. Also as a weakness of *Open3D*, there is no direct production of a TIFF format image when capturing the screen. This may lead to additional loss of quality in the image due to it requiring further processing.

The presentation of the produced images could be done much better using great image processing system for things like the scale bar, border around theses images and section and elevation lines on the floor plan/top view rather than using *matplotlib* or plotting them manually.

# ACKNOWLEDGMENTS

The work done in this project was not done without input from other people. I would like to show my gratitude to my supervisor, Patrick Marais, and my project partner, Claudious Nhemwa for their contributions and insight. Additionally, this is extended out to the Zamani Project for providing data.

# REFERENCES

[1] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G., et al. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference* (2008), vol. 2008, Salerno, Italy, pp. 129–136.

[2] Gregori, R. M., Volpato, N., Minetto, R., and Da Silva, M. V. Slicing triangle meshes: An asymptotically optimal algorithm. In *2014 14th International Conference on Computational Science and Its Applications* (2014), IEEE, pp. 252–255.

[3] McMains, S., and Séquin, C. A coherent sweep plane slicer for layered manufacturing. In *Proceedings of the fifth ACM symposium on Solid modeling and applications* (1999), pp. 285–295.

[4] Oliphant, T. E. *A guide to NumPy*, vol. 1. Trelgol Publishing USA, 2006.

[5] Roberts, L. G. Homogeneous matrix representation and manipulation of n-dimensional constructs. *MIT Lincoln Laboratory, Technical Report, MS1405* (1965).

[6] Schmidt, J. M. Interval stabbing problems in small integer ranges. In *International Symposium on Algorithms and Computation* (2009), Springer, pp. 163–172.

[7] Schwarz, M., and Seidel, H.-P. Fast parallel surface and solid voxelization on gpus. *ACM transactions on graphics (TOG) 29*, 6 (2010), 1–10.

[8] Sproull, R. F., and Newman, W. M. Principles of interactive computer graphics. *International Edition* (1979), 3–47.

[9] Sutherland, I. E., and Hodgman, G. W. Reentrant polygon clipping. *Communications of the ACM 17*, 1 (1974), 32–42.

[10] Tata, K., Fadel, G., Bagchi, A., and Aziz, N. Efficient slicing for layered manufacturing. *Rapid Prototyping Journal* (1998).

[11] Van Kreveld, M., Schwarzkopf, O., de Berg, M., and Overmars, M. *Computational geometry: Algorithms and Applications*, 3rd ed. Springer-Verlos TELOS, 2008.
[12] Wang, D.-X., Guo, D.-M., Jia, Z.-Y., and Leng, H.-W. Slicing of cad models in color stl format. *Computers in industry 57*, 1 (2006), 3–10.
[13] Zhou, Q.-Y., Park, J., and Koltun, V. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847* (2018).

# Appendix

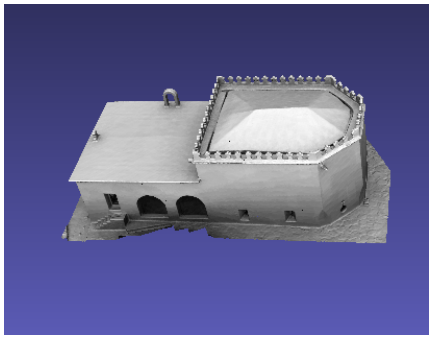
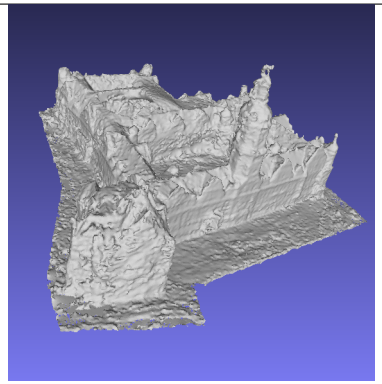
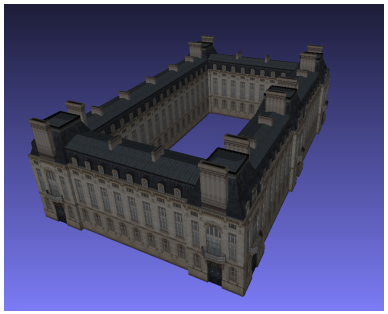
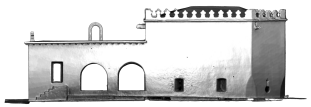
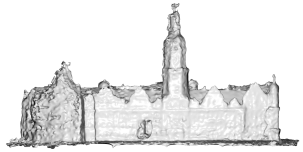
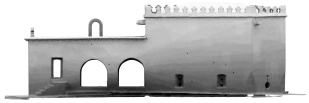
| Image type | Chapel of nossa senhora de baluarte | Green Vault Museum | Paris Police Station |
|---|---|---|---|
| Real life images |  |  |  |
| Meshlab Snap-shots |  |  |  |
| Elevations |  |  |  |
| No vertex normals |  |  |  |

**Table 2: Types of images of the three-dimensional models**

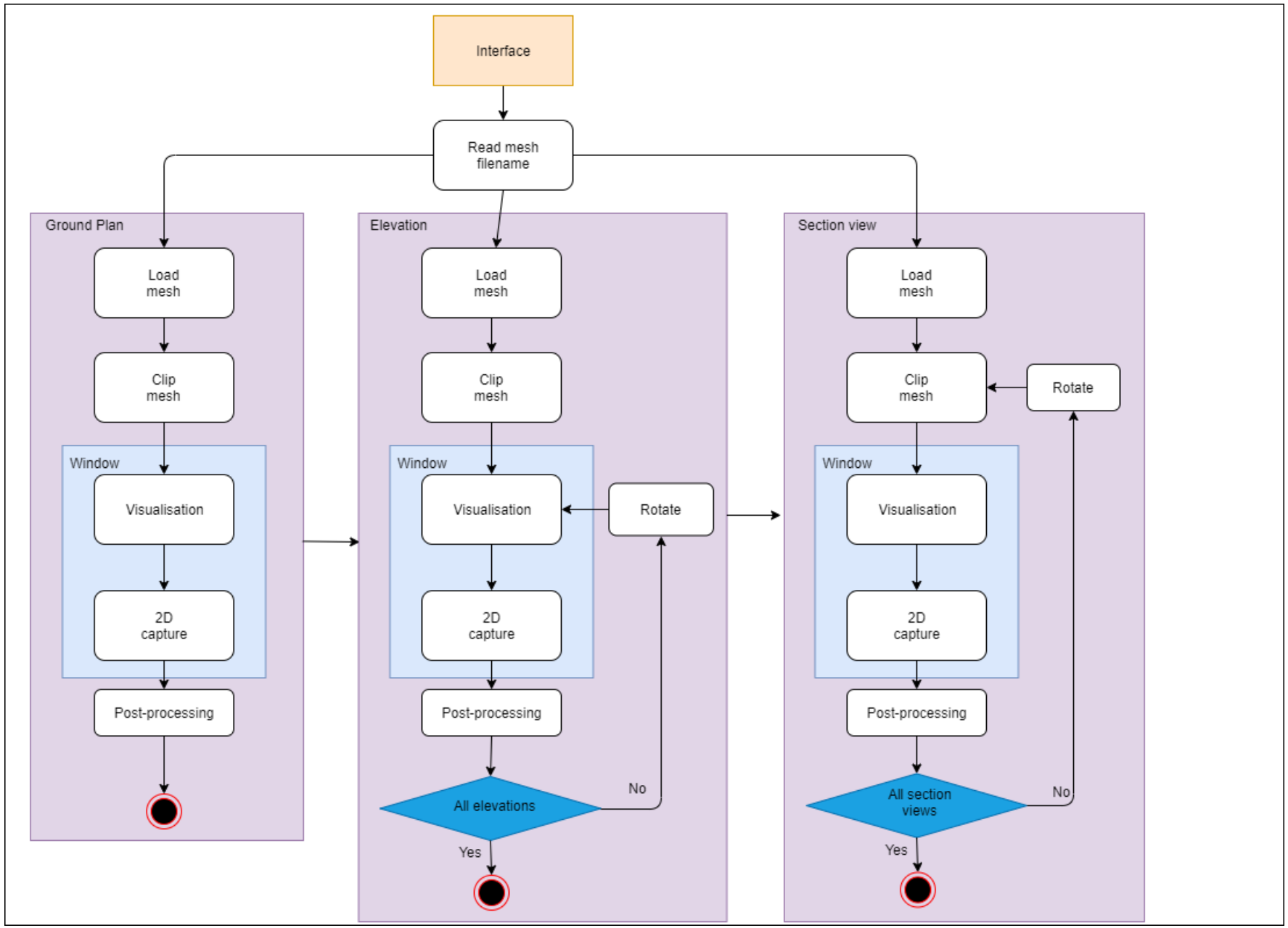



**Figure 20: Architecture diagram of the main processing and the post-processing**

## Images of the Chapel

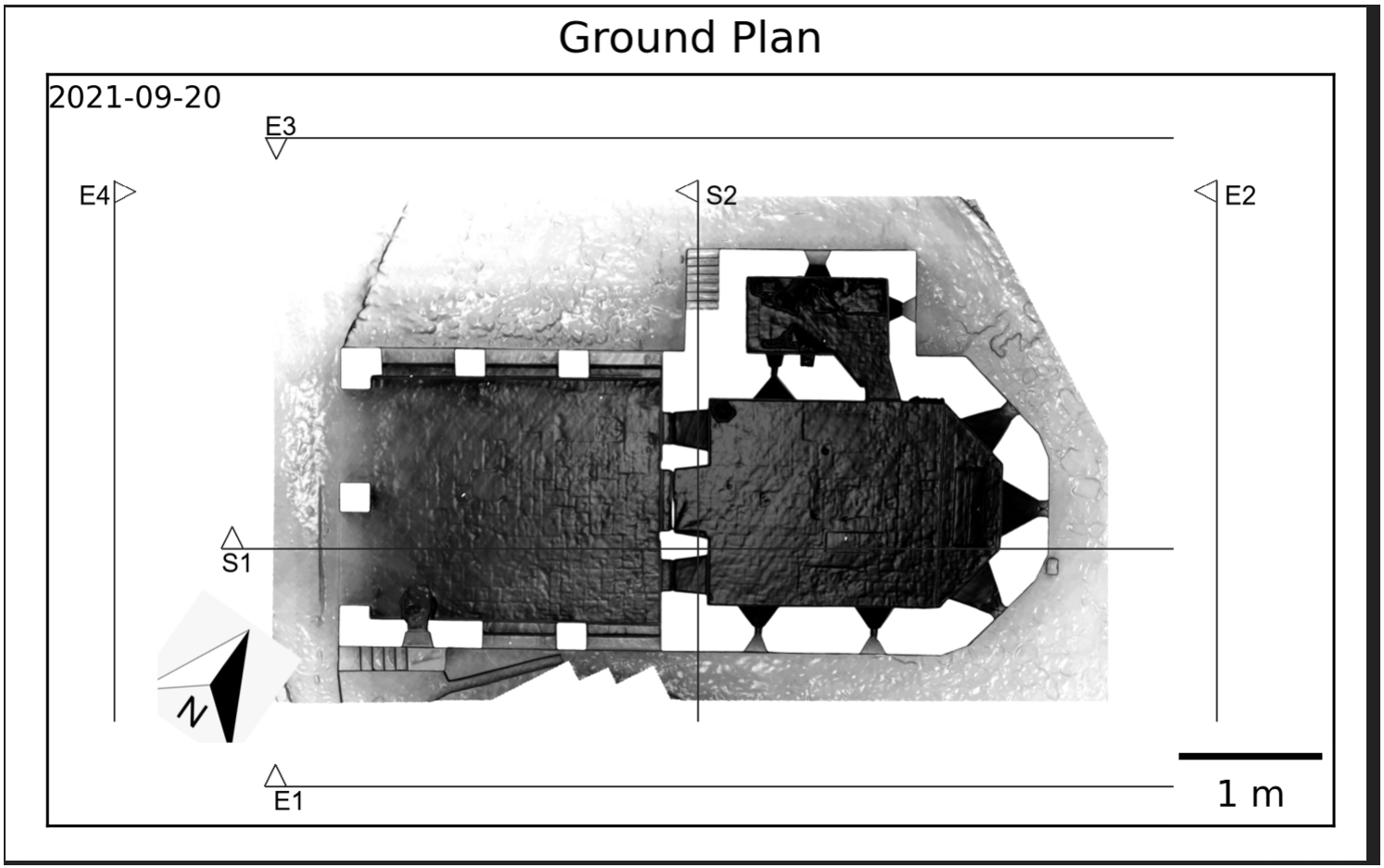



**Figure 21**

# Elevation E1

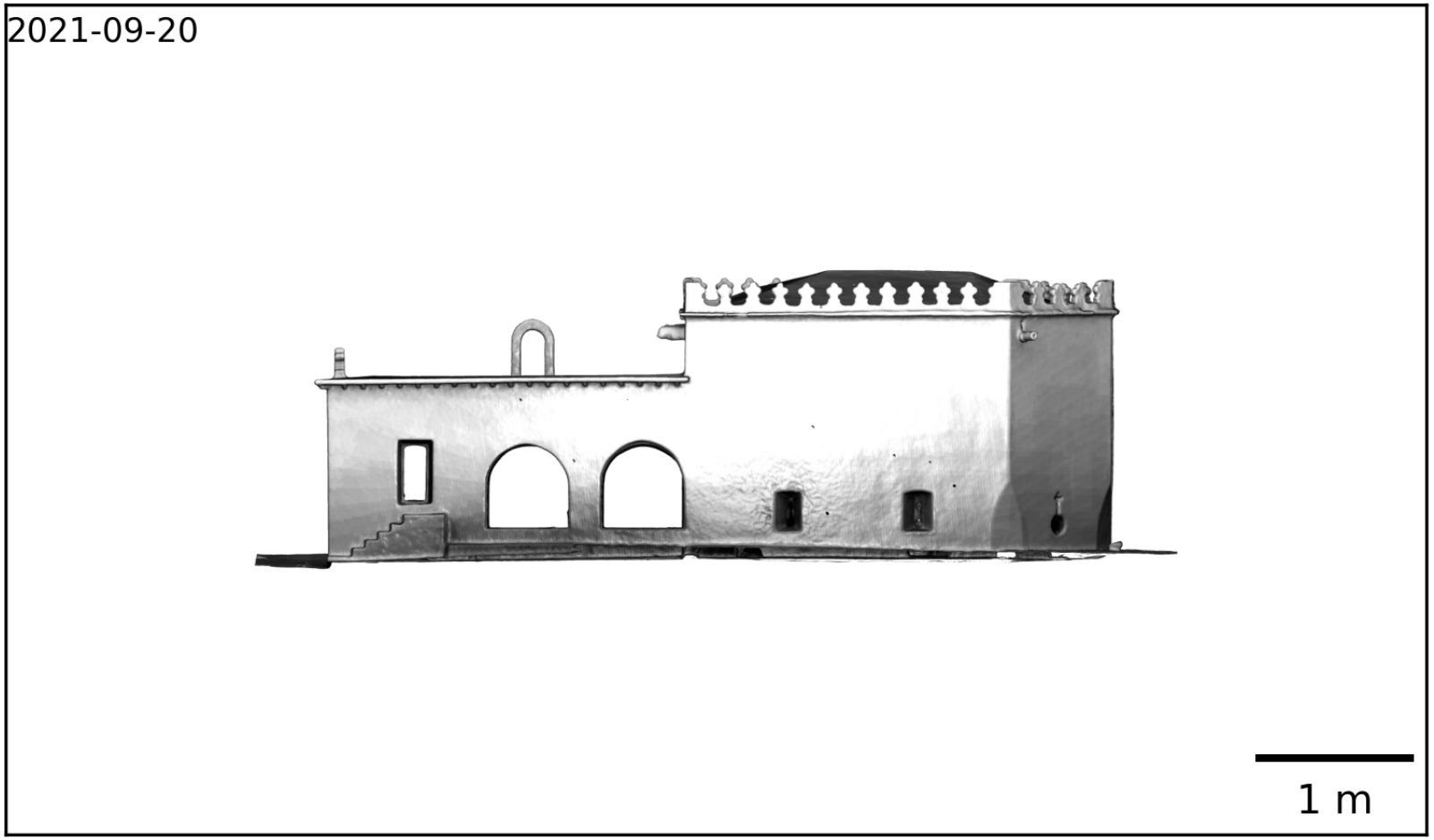
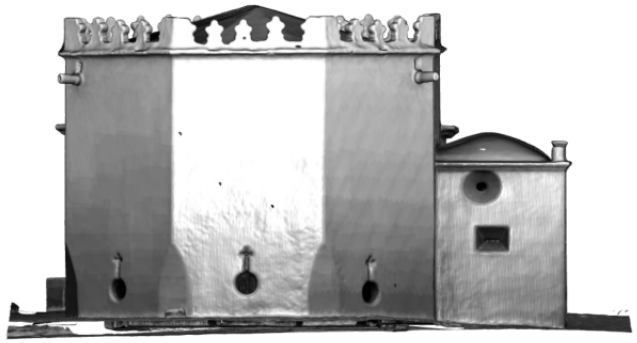





**Figure 22**

# Elevation E2

2021-09-20



1 m

**Figure 23**

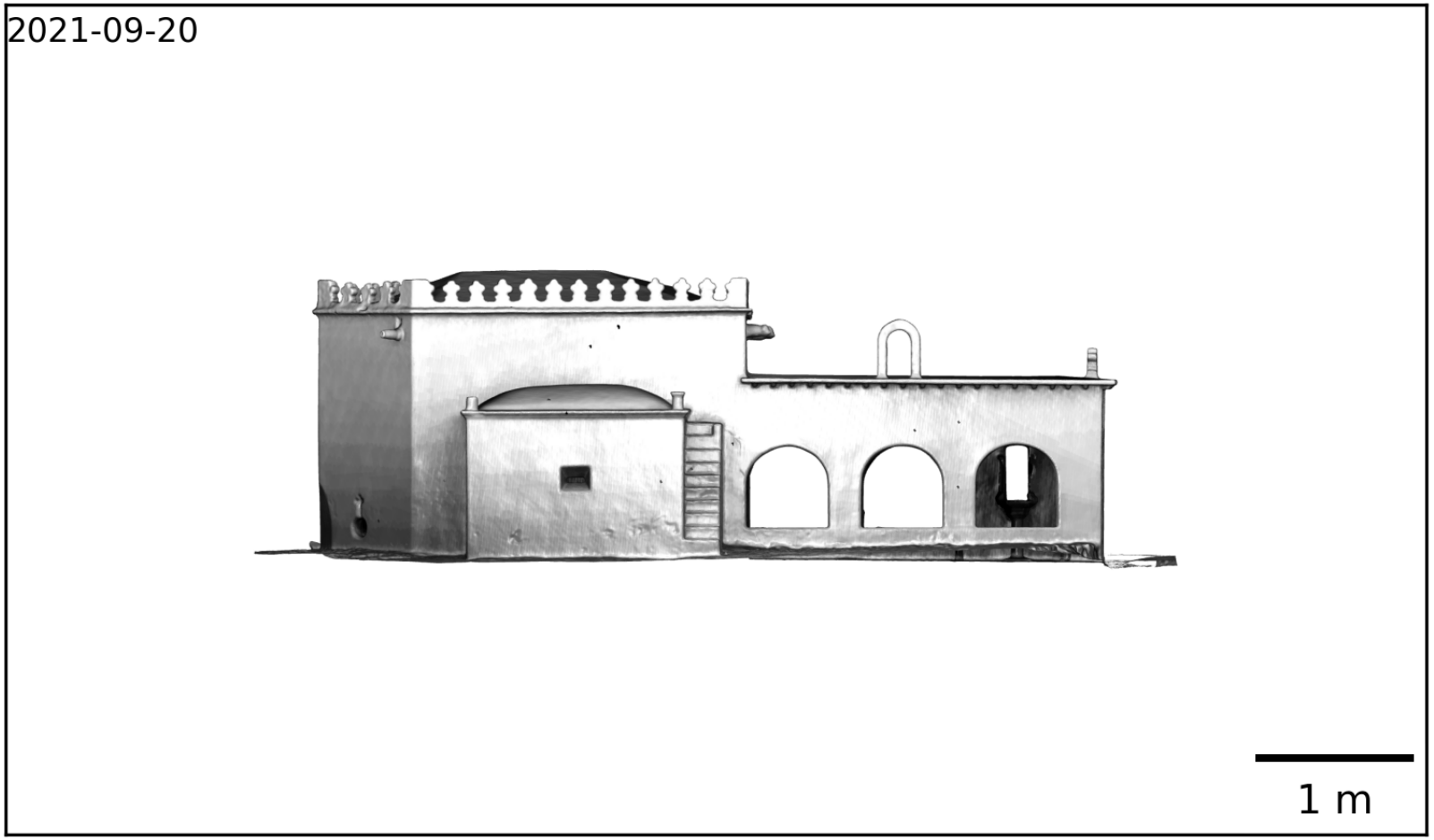



**Figure 24**

# Elevation E4

2021-09-20

1 m



**Figure 25**

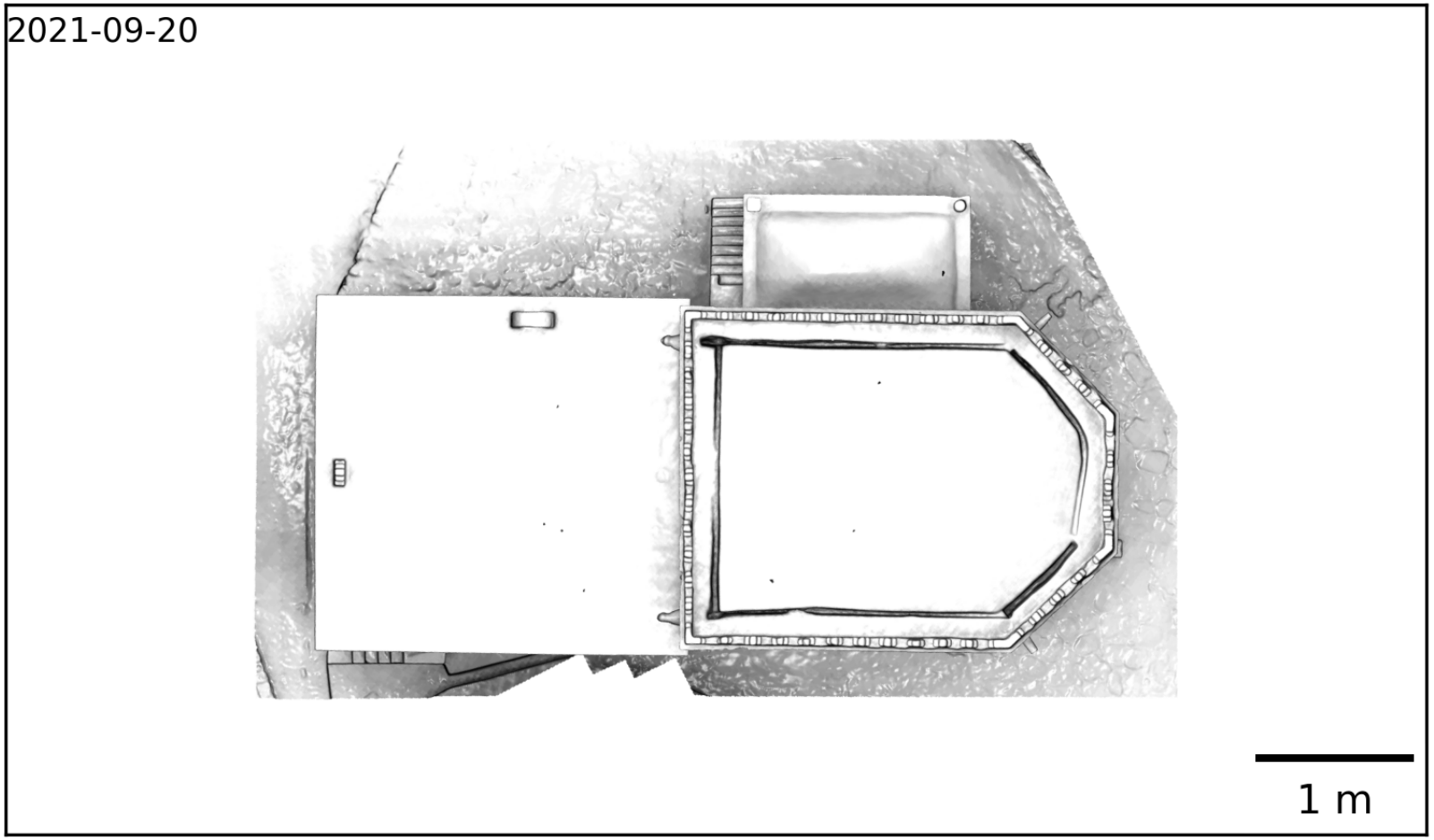



**Figure 26**

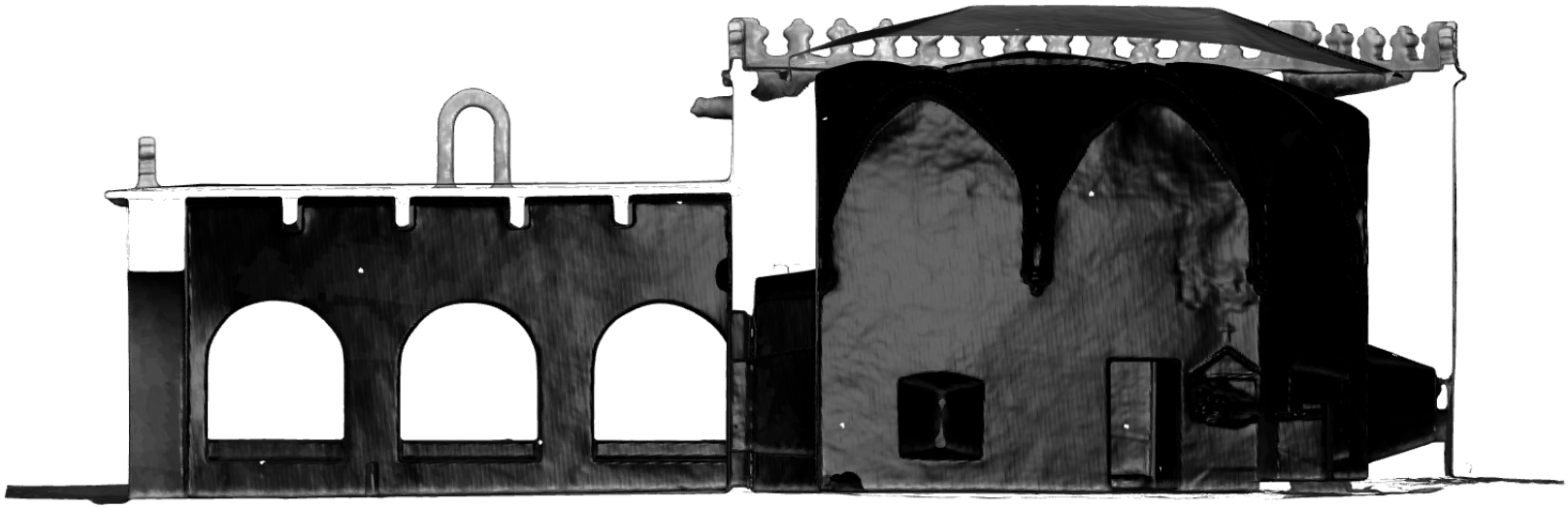


**Figure 27**

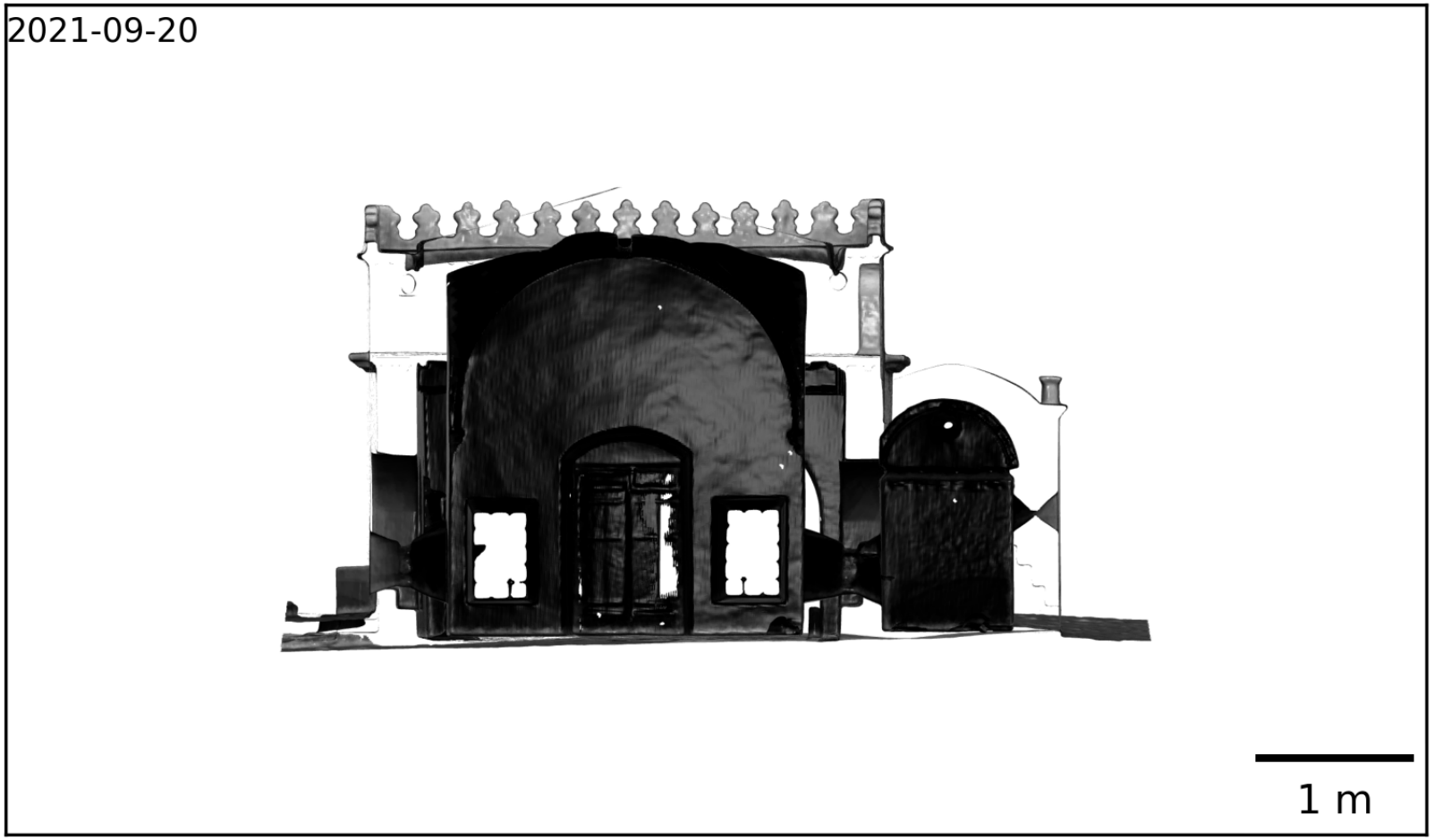



**Figure 28**