Maya Archaeology Reconstruction: From Scan to Tour

KOLIN GUO*, SHRUTHEESH RAMAN IYER*, and TOMMY SHARKEY*, UC San Diego, USA

There's an ongoing tension between the preservation of cultural heritage sites and the need for usable land. As a result, there have been several efforts to preserve these sites using current advances in scanning, modeling, and visualization technologies - notably RGB-D cameras, scene reconstruction pipeline, and Virtual Reality (VR). However, these individual technologies have mostly been developed independently, and little effort has been dedicated to integrating them. This paper presents an evolution of an existing system for reconstructing and displaying cultural heritage sites in Virtual Reality environments. To achieve this, we develop a pipeline to (i) track the cameras using visual-inertial SLAM, (ii) perform a 3D reconstruction using registered depth and RGB data, (iii) facilitate loading and displaying the reconstruction in VR, and (iv) create virtual voice-guided tours. We discuss the details of this automated pipeline and demonstrate its ability to take an unskilled user from data capture to an immersive virtual tour without the need for 3rd party or command-line tools by designing a desktop GUI interface. We hope that this automated scanning and reconstruction pipeline will help the digitization and education of cultural heritage sites.

ACM Reference Format:
Kolin Guo, Shrutheesh Raman Iyer, and Tommy Sharkey. "Maya Archaeology Reconstruction: From Scan to Tour." *CSE 237D Embedded Systems Design Project, March–June, 2022, UC San Diego, La Jolla, CA.*

1 INTRODUCTION

Archeologists today document their findings by hand drawings and sculptures. This is also true for the archeologists who are exploring the extensive network of tunnels and caves of the Maya Civilization. However, these hand documentations are extremely time-consuming and are also prone to errors on a global scale. To preserve these sites, we propose to build on prior work that attempts to tackle this problem by creating a digital scan of the location. This scan or "reconstruction" is then handed off to a Virtual Reality (VR) application, which enables users to immerse themselves, walk around, and explore the site. We aim to update the scanning hardware to a newer and lighter RGBD + IMU camera system, provide real-time feedback on the scanning process, use an offline algorithm to generate a high-quality scene model from the scan, form a simplified pipeline that brings this model into the VR application, and allow for the creation of audio snippets/tours.

We thus create a pipeline that fills the gaps between 3D scanning, reconstruction, and virtual reality technologies, to facilitate an unskilled user in scanning, reconstructing, and immersing an audience in a digital replica of a cultural heritage site. Through this system, the user would first scan the area using an Intel Realsense D435i connected to a laptop with visualization which would provide real-time visual feedback, build a 3D model of the scan offline, and finally load the model into VR, all through an automated pipeline with minimum manual inputs.

This work builds on previous work that investigated the effectiveness of low-cost technology in creating digital reconstructions [7]. This system used two Microsoft Kinect v1, a tablet, and a laptop computer to scan archaeological sites. The laptop would sit in a backpack, processing data from the camera. The cameras and tablet were affixed to a handheld rig along with an LED light panel. The camera was plugged into the laptop as well as a power source via two cables, and the tablet was similarly plugged in via a third cable (see Figure 1). A command-line interface was used to initiate the program to begin capturing and processing data.

*Authors are alphabetically ordered by first names. GitLab repository for our code: https://gitlab.nrp-nautilus.io/tsharkey/maya-archeology-unity

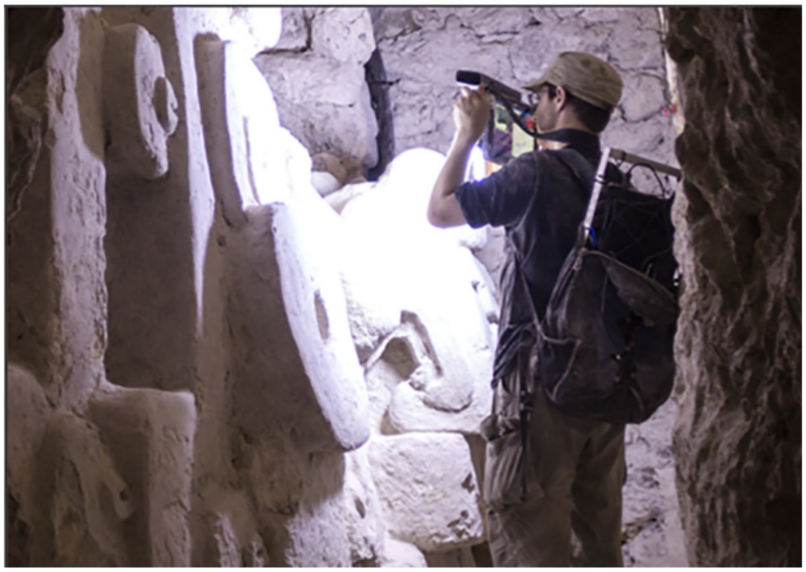


Fig. 1. The original scanning system with several cables leading into the user's backpack in a cumbersome setup.

The scanning process would produce a point cloud that would then be refined into a usable file in virtual reality. The point cloud would be loaded into MeshLab[1], an open-source mesh processing software, where it would be converted into a mesh or a 3D model. This mesh would be exported and then imported into ZBrush[2] where it would be trimmed and edited by hand to fix any artifacts from creating the mesh. Once done, the mesh would be cut into several smaller pieces to allow for optimized rendering in VR.

The pieces of the mesh would each be brought into Unity[3] so they could be loaded onto the VR headset. Inside Unity, a new "scene" or game level would be created and each mesh would be positioned in that level relative to the others. The player controls would be re-imported into the created scene and placed on the combined mesh. Additionally, audio recordings explaining the various components of the scan would be created, added to the project, and hard-coded into a list of audio files. Extra assets (*e.g.*, bowls, animated spiders, and monkeys) would then be added to finish off the scene. The menu system would then be manually updated by the programmer so the scene could be loaded, and then the entire application with the mesh assets would be built and deployed.

However, there are a few issues with the existing system, both in terms of accuracy of geometry reconstruction and complexity of usage. For creating a 3D model, the system makes use of photogrammetry techniques collected with multiple cameras and viewpoints, combining multiple independent images into a 3D model. To produce an accurate 3D model, this technique involves a lot of manual work and has multiple pre- and post-processing steps (such as stitching close-by images together). Due to this, it is not extremely feasible to use photogrammetry with videos taken from moving cameras. In addition, the VR environment creation also involves multiple manual steps before being deployed.

We aim to reduce the complexity of this system by updating the camera hardware and forming an automated GUI-based pipeline to hand off between each of these steps, while also focusing on improving the accuracy of the

[1]https://www.meshlab.net/
[2]https://pixologic.com/
[3]https://unity.com/

resulting reconstructed meshes. We replace the Kinect with a USB-powered Intel RealSense D435i camera which should give better color and depth measurements. The GUI handles the scanning process, the SLAM algorithm, the mesh reconstruction step, and the final mesh splitting step. The resulting meshes are ready to be loaded into VR at runtime, and the interface also allows for the addition of all extra assets at runtime. In doing so, the system shifts from requiring individuals with programming and modeling expertise to an integrated system that doesn't require any technical expertise to use. We replace photogrammetry methods with a Simultaneous Localization and Mapping (SLAM) algorithm, making use of temporal information as well as the inertial measurement unit (IMU) inside the D435i camera.

1.1 Contributions

Through this work we aim to contribute to prior work in cultural heritage digitization via our system as well as a cursory evaluation and discussion of how the pipeline it creates helps to democratize the digitization of cultural heritage sites.

2 RELATED WORK

2.1 Cultural Heritage Reconstruction

While on initial inspection, the digitization of cultural heritage sites seems to be an innately positive preservation of our history; the specialized skills and resources needed for digitization lead to problems around site selection. Manzuch et al. describe the selection process around which cultural heritage sites do and don't get reconstructed as being controlled by Western[4] nations [13]. They look at the drive-in communities like various Native American Indian communities, motivating the digitization of cultural heritage. But Manzuch et al. find that in practice, the sites that get preserved are the already well-preserved sites coming from a Western European ancestry. They point to the expertise required for digitization and call for reforms around who gets to choose which sites are digitized and how those sites are shared.

The required expertise for digital reconstruction is a result of the complex algorithms and tools needed for scanning and constructing the data into a 3D model. While there are several alternative camera systems and algorithms [17, 18], most require expensive camera systems or even drones [27] and site-specific mounting hardware [4]. While the operation of this equipment and developer requirements place strain on the digitization process, a questionnaire created by Adane et al. found that teams working on the digitization of cultural heritage sites most commonly face problems around managing the digital assets they created [1]. Teams described problems with the creation of tours, distribution of digital assets, and the automation of the entire process.

2.2 SLAM and Dense 3D Reconstruction

[18] provides an overview of the different methods used for 3D reconstruction. Of these, Stereo Photogrammetry, and time of flight based camera methods are the only ones that can also provide color information. Most of the techniques for 3D reconstructions make use of photogrammetry. Agarwal et al.[2] takes hundreds of images, and uses photogrammetry to reconstruct the City of Rome. While these methods have since evolved and have demonstrated impressive results [9], they generally work well only with aerial photogrammetry (i.e. images captured from drones), and do not translate well with hand-held cameras where global context is missing. Another line of research thus focuses on using Simultaneous Localization and Mapping (SLAM), a technique that can track cameras in real-time and build a map of the environment at the same time. Since this is performed at real-time it also provides the added benefit of visual real-time feedback

[4]"Western" referring primarily to Europe, the United States, and Canada

during the 3D scanning process. Visual SLAM (i.e. SLAM with cameras) provides localization information of the cameras, which are important, since there are no global positioning system inside caves.

Visual SLAM is an old yet relevant problem in robotics that has been investigated since the 1970s. There are two main approaches to solve the visual SLAM problem; feature based and direct methods. Feature based methods use key-points in images, to track them across frames, save these into local submaps and then run a graph optimization or bundle adjustment [22], while dense SLAM methods directly use pixel intensities of images and use photometric loss for error estimation and optimization. In addition, to improve localization accuracy, modern SLAM algorithms also use an inertial measurement unit (IMU). The IMUs provide additional tracking information with very high frequency, that can be combined locally with low frequency image stream.

Direct SLAM methods can simultaneously create a dense reconstruction of the environment as part of the mapping thread. DTAM [16] was the first method using direct SLAM, and was used to reconstruct an office space with good accuracy. Since then several methods have built upon them, such as ElasticSLAM [25], LSD-SLAM [6] and DSO [5]. However, dense SLAM methods are typically memory intensive, and do not work well over long distances.

Feature based (or sparse) SLAM methods on the other hand first obtain the camera trajectory (with a sparse map), and then use backprojection 3D reconstruction techniques using the camera trajectory. These methods focus on obtaining accurate camera poses (position and orientation). ORB-SLAM [14], and its successor ORB-SLAM2 [15] are the most popular and mature SLAM implementations, that track ORB features across frames, and perform local graph optimization, and global loop closure. However, feature based SLAM methods tend to lose track in feature-less spaces (which is expected for the Maya sites). In order to solve this, many IMU based solutions have been proposed towards sparse SLAM. Using IMUs, [20], [24], [3] and [8] have all shown impressive performance in "difficult" featureless environments, thus highlighting the benefit of using an IMU. ORBSLAM3 [3] uses ORB features to track, and provides a map management system, Basalt VIO [24] and OpenVins [8] use FAST features, while maplab [20] uses a patch descriptor. Since the proposed solution makes use of an Intel Realsense D435i camera that is equipped with an inertial sensor, the IMU-based VSLAM approaches were used for this work. In addition, since Basalt VIO requires stereo images, it was deemed unsuitable for this camera.

Once the camera trajectory is estimated, we perform a dense 3D reconstruction using the RGB-D observations to generate a triangular mesh of the scanned site. Almost all popular methods for reconstructing a large environment from RGB-D images are based on the truncated signed distance function (TSDF) introduced in the KinectFusion paper [16]. TSDF-based approaches divide the 3D scene volume into small voxel grids, each storing an observation weight and a truncated distance value to the closest mesh surface. Due to its simple voxel update step, TSDF-based approaches can integrate hundreds of RGB-D frames in under one minute. The observation weights with their moving average updates are also great in addressing the Gaussian noise along the surface normal and producing a smooth surface output. To make the scene volume more memory efficient, a scalable TSDF volume based on spatial hashing is introduced to reduce memory consumption [28]. From a constructed TSDF volume, a triangular mesh can be extracted by running the marching-cubes algorithm [12]. In our experiments, we tested two sets of TSDF-based algorithms: OpenChisel [10] and an Open3D-based TSDF volume integration [28, 29].

2.3 Virtual Reality

Lasala et al. describe their own process of reconstructing the Timpanogos Cave System and their difficulties [11]. They describe the capability of modern game engines for rendering extremely high fidelity graphics in reconstructed geometry. They cite optimizations in occlusion and other strategies used by game engines to reduce the number of triangles[5] rendered by the GPU. In order for these game engines to take advantage of these tricks, however, the mesh needs to undergo some level of pre-processing and formatting. This process normally consists of texture compression and mesh optimization. These processes allow the GPU to access mesh data, offloading mesh processing to the GPU. When importing a 3D model file (.obj, .fbx, .ply, etc.), the models are converted into a form the game engine can readily send to the GPU. Game engines typically provide tools for importing models that perform this task for the developer. This becomes a problem when those tools are only made available through the game engine's editor, and aren't available in a deployed app at runtime. Because of this, the runtime import of 3D models requires custom parsers - complicating the picture that Lasala et al. present.

Additionally, one of the most effective 'tricks' Lasala et al. describe is Occlusion Culling. This is the process where a render engine determines what elements it needs to render (what elements are in the user's field of view) and what elements are either out of the field of view (e.g. behind the user) or are occluded by other objects. The occlusion process works by creating a mask from each rendered element and raycasting the edges of that mask out from the camera. Assets are culled (removed) from the render queue based on whether or not they fall in this mask. The important feature of occlusion masks to note is that they only work on independent models - a single enormous model that has triangles behind the user or that self-occludes will typically still be rendered in full. When considering models of cultural heritage sites, their size can potentially overwhelm a rendering engine without being broken into pieces to allow for occlusion culling. This process is not handled by current game engines and is often done by hand. A 3d modeller will open the file and cut it into several pieces, exporting each piece so it can then be imported into the game engine. Once imported, the pieces are manually re-aligned into an asset that can be viewed via the game engine.

3 SYSTEM DESCRIPTION

Using an RGB-D camera with IMU and a VR headset, we developed a pipeline to facilitate an unskilled user in scanning an indoor site, densely reconstructing the scan, and finally exploring the scanned site in a VR environment. When creating the pipeline, we emphasized the interoperability of the modules, allowing an effortless hand-off between each of the steps. In this way, we can better automate and integrate the entire process, reducing the need for expertise in the fields of modeling, VR, and scanning.

The process of going from data capture using the D435i camera to a 3D mesh model is broken down into two steps: a real-time visual-inertial SLAM to obtain camera trajectory, followed by a dense 3D reconstruction. The resulting 3D model is then passed onto VR for visualization. To ensure modular codebase and multi-platform support, multiple docker containers are created, each running a step in the pipeline. As a result, one container initializes the camera, and streams the data, a second container runs real-time visual inertial SLAM, and finally another container performs 3D reconstruction and mesh splitting.

3.1 Intel Realsense Interface

This is a small module that opens the Realsense D435i camera that is connected to the System via USB, streams the RGB, depth images and the IMU data as messages through the Robot Operating System (ROS). Support is provided for streaming at different resolutions (848x480, 1280x720). A small display of the RGB stream is presented to the user for visual feedback to the user.

[5]"triangles" are pieces of a 3D mesh. Modern GPUs have been optimized toward the rendering of triangles, and "number of triangles" is often used as a metric for rendering graphics strain and performance.

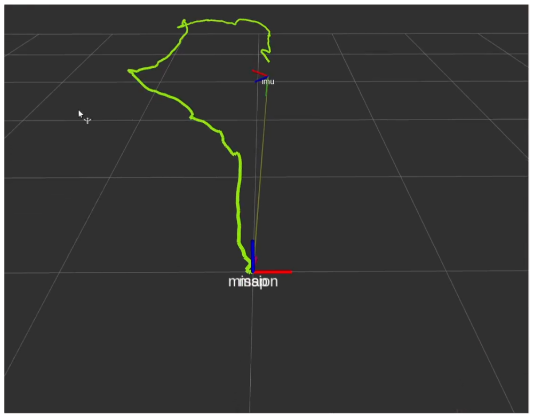


Fig. 2. Camera Tracking using SLAM

3.2 Real-time Visual-Inertial SLAM

The goal of real-time SLAM is to track the camera trajectory accurately, in real-time. Accuracy of the camera pose and real-time operation are the key criteria for consideration for this module. This module is interfaced with ROS as well. Since IMUs can improve the performance of SLAM, and since the Intel Realsense D435i provides IMU support, Visual Inertial SLAM algorithms are used for SLAM. Maplab [20] and ORBSLAM3 [3] are implemented, and interfaced with the realsense stream. Once the module is started, a display opens that shows tracking information to the user in two windows. The first window displays the trajectory of the cameras from the start, while the second window shows the feature tracks used by the sparse SLAM. A visual feedback in this second window informs the user if tracking is lost, indicating that the user must slow down or go back to a previous position and restart tracking from that position. Figure 2 shows a snapshot of the real-time SLAM tracking process.

3.3 Dense 3D Reconstruction

Given the estimated camera pose trajectory from the visual-inertial SLAM algorithm, dense 3D reconstruction is performed to generate a PLY mesh for the entire scanned site. For this part, two sets of TSDF-based algorithms are tested: OpenChisel [10] and an Open3D-based TSDF volume integration [28, 29].

OpenChisel. OpenChisel is an implementation of a generic TSDF 3D mapping library based on the Chisel mapping framework [10]. In our experiments, we found that it does not function properly when the TSDF voxel resolution is below 0.04 meter. We think this is because OpenChisel is originally developed for running on limited hardware platforms which do not have enough memory resources to run mesh reconstruction at millimeter resolution. Therefore, due to its presumably suboptimal implementation and our goal for high-detailed geometry reconstruction, we do not recommend using OpenChisel for dense 3D reconstruction in this project.

Open3D TSDF with Pose Refinement. The increasingly popular 3D geometry processing Python library, Open3D [29], contains several pipelines for dense 3D reconstruction given a known camera pose trajectory. We have tested its tensor-based dense RGBD SLAM integration and its tensor-based VoxelBlockGrid integration. Both always crash the kernel when extracting the triangular mesh with the marching cubes algorithm. Therefore, we switched to its

ScalableTSDFVolume integration which we observed stable performance with accurate high-detailed geometric reconstruction.

To further improve the estimated camera pose trajectory, we included an additional camera pose refinement step between consecutive camera frames. The pose refinement step is based on a multi-scale iterative closest point (ICP) algorithm using both point-to-plane loss and color intensity loss. The reconstructed mesh is then split into chunks using PyVista [23] for efficient loading into a VR environment. We chose the PyVista library because other 3D mesh processing libraries (*e.g.*, trimesh and Open3D) have not yet correctly implemented texture interpolation during mesh clipping.

3.4 Virtual Reality

The Virtual Reality component of the system is built on the Unity game engine and uses the Oculus Integration package to target the Oculus Quest and Rift headset lines (the most popular VR headsets as of 2022). Site data from the reconstruction is saved in the device's filing system and the TriLib package is used to load the 3D models at runtime. The JSON file from the reconstruction is loaded and filled with extra data about the positioning of various assets. This file is used to save and load the site between successive experiences. Due to a discrepancy between Open3D's output texture format (vertex colors) and Unity's native format (texture map), a simple vertex shader is used to ensure colors can be accurately seen in VR.

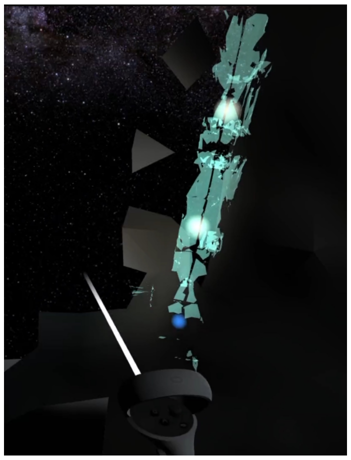
When a new site is added to the filing system, the application will use the JSON configuration file to recognize the new files and organize them into a new directory. Once there, each model is loaded into the environment. Models are loaded via a series of threads due to their potentially large size freezing the VR experience for the user. Threading also allows the system to provide progress feedback on the loading process (see Fig. 3c). Once loaded the models appear about the origin of the space, positioned relative to each other based on the JSON configuration file's specifications. A user with an admin password is then able to unlock features used to reposition the site and setup the tour experience.

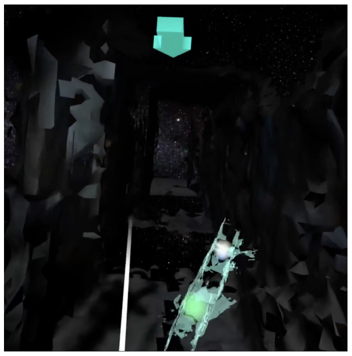
Once in admin mode, the user can use their controllers to reach out and grab the reconstructed mesh. Grabbing the mesh allows it to be repositioned so that a user looking to visit the digital reconstruction will begin in the correct location (e.g. at the entrance of a tunnel). Once positioned, the JSON file is updated to reflect the transformation of the entire site from the origin to the target location (the relative poses of each mesh remain the same).

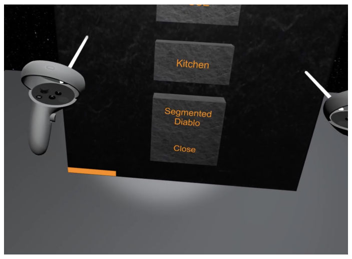
Once complete, the user can walk around, holding a button on their controller to record audio snippets. These snippets can be located near interesting or noteworthy elements of the heritage site and record tour-like explanations and highlights of what a user is looking at as they explore the site. The admin user can press a different button to spawn various objects in the space. Currently, the system supports spiders and howler monkeys which animate and make sounds. These are used to enhance the experience of being in a particular environment, and the list of placeable items can be expanded to include more animals or objects like tools.

Once setup, the admin user can exit the admin mode or close the app before giving the headset to another user. That user, when they open the application will be able to choose from any of the sites on the headset, including the one the admin user setup. Selecting the site will load and position the models, audio, and extra objects/animals. This system was specifically designed to be used in for cave reconstructions, so the user has a head mounted virtual light source, so when they look around the site, only the space in front of them is illuminated.

Users move through the space via 3 different methods. The first is simply through their physical movements walking around their space. As many sites are far larger than the physical space a user may have access to, the joystick can also be used to move around the space. Additionally, the user can teleport around the space to more quickly reach far away


(a) The minimap showing the glowing blue location of the user and glowing white location of audio recordings in the site


(b) Arrow guiding user to selected audio point (glowing green on the minimap)



(c) When loading a site, an orange progress bar appears

Fig. 3. Virtual Reality experience

places. The movement controller in Unity is modified to add additional collision behaviors to prevent the user from walking through walls. This was specifically done so that in a narrow cave environment, the user is forced to squeeze through tight areas and duck their head under low ceilings - improving their sense of presence in the space.

Additionally, the user is given a minimap (see Fig. 3a)on their wrist that can be toggled on and off. When shown, the map is a transparent scaled down replica of the site with a glowing blue indicator showing where the user currently is. This can be used to help guide the user through more complex sites (e.g. a cave system). The map has additional glowing regions that can be selected indicating where audio recordings have been created in the site. When selected, an arrow appears in front of the user guiding them toward the point that they selected (see Fig. 3b).

As a result of these components, the Minimum Viable Product (MVP), of a pipeline that can allow a user to scan a site and view it in VR without having to write any code has been completed, with additional features.

4 EVALUATION

We evaluate our system through a self critique of our ability to meet our user story and individual evaluations of the various components and how well they did their job. All of this is subjective self-critique.

To guide our system we created a user story that consists of an example user with a backpack and laptop. A RealSense camera is fixed to a tablet, and that camera is plugged into the laptop. The tablet shows visual feedback of the data being collected by the RealSense camera, through which the user can see where they are missing data and when they lose and regain tracking. The user can spend several hours scanning, and when done, they leave the site. Once home, they plug in their laptop and press a button on the tablet/laptop to begin creating a model from the raw data, and leave it to run overnight. The next morning they have a new directory with several .ply files in it. They plug their VR headset into the computer, and drag and drop those files into the headset. Removing the headset from the computer, they open the viewer application on the headset and see an interface to open a saved site. They can see previous sites they have visited, as well as the site they just loaded the data for. Opening the new site loads their newly created model, allowing them to walk around the site in VR and explore.

To facilitate this smooth pipeline containing just a few steps, we developed a GUI, that walks the user through the different steps required for end model. The GUI helps the user install the necessary docker containers, and prompts user for specific inputs at each step, as shown in 4

4.1 Real-time scanning feedback

We performed two of these walk-through tests, using the iteration to improve the scanning GUI. Because of the potential for creating multiple scans and needing to save multiple files, we found that we needed inputs for the user to save files to various locations. Additionally, because the offline reconstruction can be time consuming we wanted users to have control over the quality of the reconstruction. As a result we needed to add some input parameters to our original GUI (see Fig 4). We also added the stdout data from the installation and process scripts.

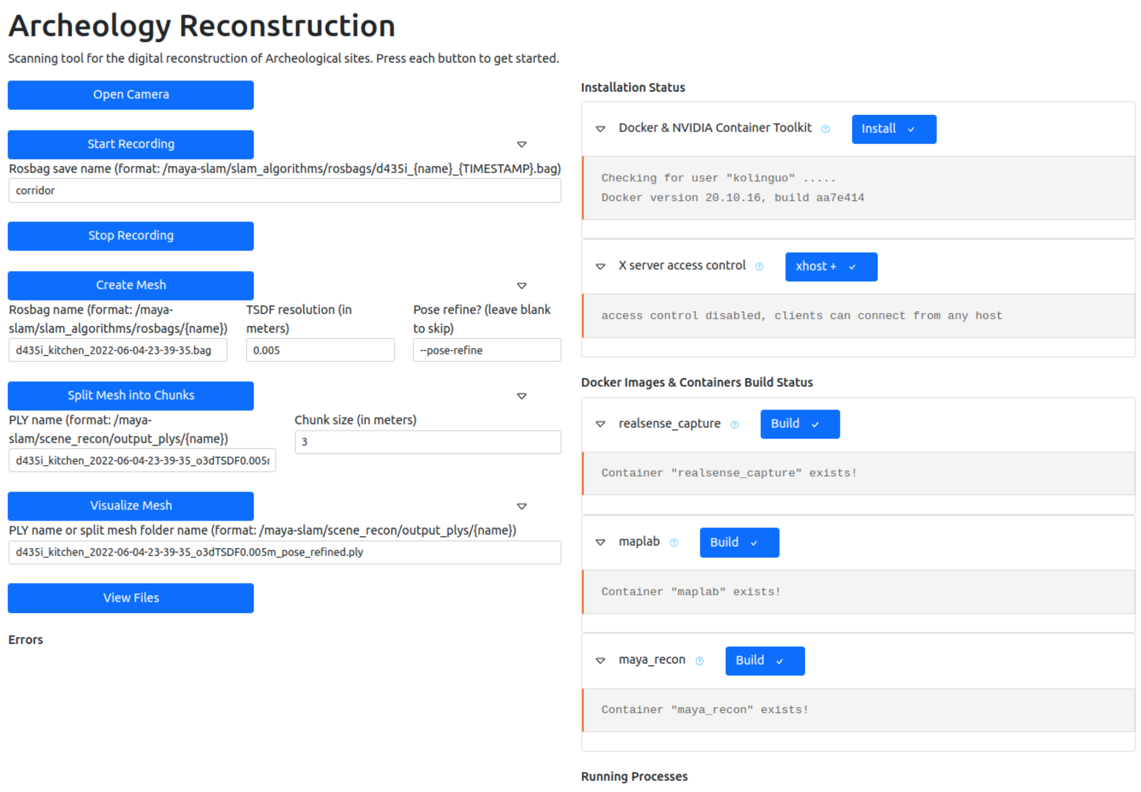
To avoid cluttering the interface, the additional elements (parameters and stdout) are hidden initially. Each process and installation step is initially presented simply as a button. A dropdown arrow is placed next to each item that has more data hidden beneath it. Opening the dropdown reveals the additions to the interface. This served a nice balance between keeping the flow of what the user needed to do, while not limiting the interface capability and feedback to the user. Adding to this flow, the process buttons are disabled until the installation scripts all successfully complete.

To mimic the tunnel followed by a cave/room type archeological site in the Maya Caves, we collect data of the CSE corridor (that serves as the tunnel system), and the CSE kitchen, which can be thought of as a site containing important artifacts, and the process is run and evaluated on this data.

Tracking feedback was provided to the user via two popup windows. The first window shows the quality of tracking of features, while the second window shows the trajectory (pose) of the camera. In our tests, we primarily found the pose visualization useful for understanding tracking and the camera tracking visual useful mostly to understand the camera's field of view and ensure a region had been captured. The tracking visual also provided indications when tracking was lost (perhaps due to quick motion), indicating that the user must slow down while tracking.

One thing we found lacking was indication of gaps in the scanning process. It would have been useful to have a visual (e.g. point cloud) showing all of the captured data. This would let the user look around in the visual to find holes that needed filling. Once a hole was identified, the camera could be pointed at the region to fill the gap.

4.1.1 Discussion on the different SLAM algorithms. ORBSLAM3 relies heavily on the IMU measurements, and initializes using the IMU sensors for further tracking. However, since scanning the sites requires slow movement of the camera, to focus on all parts of the environment, we found that the IMUs did not get excited enough to initialize the SLAM trajectory at various points, and thus ORBSLAM3 was not suitable for this task. In fact, RGB-D SLAM with ORBSLAM3 performed better than RGB-D + IMU. OpenVINS [8] was also tested with the collected data, however it was also unable to find stable initializations for the IMUs making it unsuitable for the task. Maplab on the other hand gives priority to the image data over IMUs and hence shows good performance for this task. In addition, ORBSLAM3 tends to lose track of features at various low texture regions. This is again unsuitable since we can expect large parts of the Maya sites to be featureless tunnels and caves. On the other hand, since Maplab uses image patch for feature description, using photometric information, it is able to keep track in low textured regions as well. Thus, for the reminder of this paper, the results are presented with Maplab as the underlying SLAM algorithm. Nonetheless, ORBSLAM3 works well in some cases, and support is hence provided for that as well.



Scanning tool for the digital reconstruction of Archeological sites. Press each button to get started.

Rosbag save name (format: /maya-slam/slam_algorithms/rosbags/d435i_{name}_{TIMESTAMP}.bag): corridor

Rosbag name (format: /maya-slam/slam_algorithms/rosbags/{name}): d435i_kitchen_2022-06-04-23-39-35.bag

TSDF resolution (in meters): 0.005

Pose refine? (leave blank to skip): --pose-refine

PLY name (format: /maya-slam/scene_recon/output_plys/{name}): d435i_kitchen_2022-06-04-23-39-35_o3dTSDF0.005

Chunk size (in meters): 3

PLY name or split mesh folder name (format: /maya-slam/scene_recon/output_plys/{name}): d435i_kitchen_2022-06-04-23-39-35_o3dTSDF0.005m_pose_refined.ply

Docker & NVIDIA Container Toolkit

```
Checking for user "kolinguo" .....
Docker version 20.10.16, build aa7e414
```

X server access control

```
access control disabled, clients can connect from any host
```

realsense_capture

```
Container "realsense_capture" exists!
```

maplab

```
Container "maplab" exists!
```

maya_recon

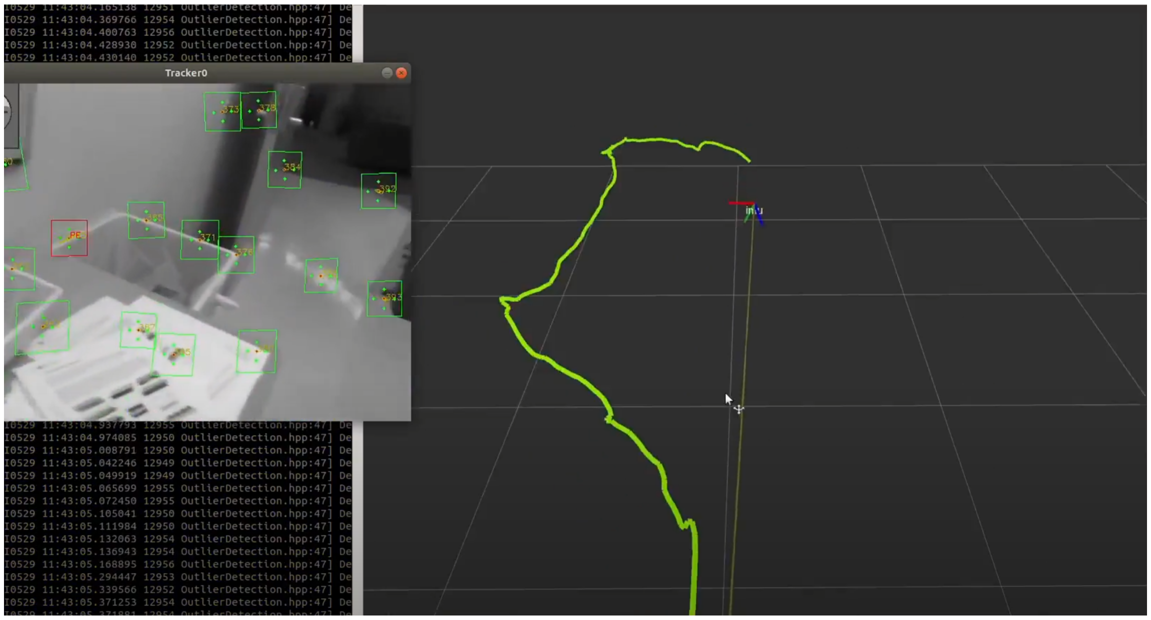
```
Container "maya_recon" exists!
```

Fig. 4. Desktop Interface GUI Example

Sequence	Total Seq. Distance (estimated) (m)	Final XYZ position (m)	Error in xy (m)
CSE Corridor (1280x720)	40	[-0.16, -0.151, 0.031]	0.22
CSE Corridor (640x480)	40	[0.158, -0.201, 0.007]	0.25
CSE Kitchen (1280x720)	25	[0.022, 0.168, -0.067]	0.169
CSE Kitchen (640x480)	25	[-0.091, 0.145, -0.097]	0.17

Table 1. Loop Error in SLAM Tracking

4.1.2 Quantative Evaluation of Real-time SLAM. Since ground truth is not available for the camera trajectory, evaluation of the accuracy of camera pose was not feasible. To quantitatively evaluate the performance, during data collection, we perform a loop, i.e. the start and end positions while tracking are the same 3D point. Thus, the error between the predicted end point and the real end point (0,0,0) is calculated, which provides an indication of the quality of tracking. This error should ideally be 0. Table 1 shows this loop error. It is clear that even over long distances, the cameras are tracked reasonable well to a high accuracy, (<1% error) which is suitable for accurate 3D reconstruction discussed in the next section.



```
I0529 11:43:04.165138 12951 OutlierDetection.hpp:47] De
I0529 11:43:04.369766 12954 OutlierDetection.hpp:47] De
I0529 11:43:04.400763 12956 OutlierDetection.hpp:47] De
I0529 11:43:04.428930 12952 OutlierDetection.hpp:47] De
I0529 11:43:04.430140 12952 OutlierDetection.hpp:47] De
```

```
I0529 11:43:04.937793 12955 OutlierDetection.hpp:47] De
I0529 11:43:04.974085 12950 OutlierDetection.hpp:47] De
I0529 11:43:05.008791 12950 OutlierDetection.hpp:47] De
I0529 11:43:05.042246 12949 OutlierDetection.hpp:47] De
I0529 11:43:05.049919 12949 OutlierDetection.hpp:47] De
I0529 11:43:05.065699 12955 OutlierDetection.hpp:47] De
I0529 11:43:05.072450 12955 OutlierDetection.hpp:47] De
I0529 11:43:05.105041 12950 OutlierDetection.hpp:47] De
I0529 11:43:05.111984 12950 OutlierDetection.hpp:47] De
I0529 11:43:05.132063 12954 OutlierDetection.hpp:47] De
I0529 11:43:05.136943 12954 OutlierDetection.hpp:47] De
I0529 11:43:05.168895 12956 OutlierDetection.hpp:47] De
I0529 11:43:05.294447 12953 OutlierDetection.hpp:47] De
I0529 11:43:05.339566 12952 OutlierDetection.hpp:47] De
I0529 11:43:05.371253 12954 OutlierDetection.hpp:47] De
I0529 11:43:05.371881 12954 OutlierDetection.hpp:47] De
```

Fig. 5. Two windows provide visual feedback while scanning - showing the features tracked by the realtime SLAM and the algorithms tracking path. Note how the pose visual (right) is detached from the path of the camera, indicating tracking uncertainty.

4.2 Reconstruction Performance

The reconstructed mesh of the CSE kitchen using the Open3D-based dense 3D reconstruction with 5mm TSDF voxel resolution is shown in Figure 7a. Two zoomed-in regions are shown in Figure 7b and Figure 7c. They show surprisingly high-detailed geometry and color that even the printed texts and whiteboard drawings are legible. For the additional pose refinement step shown in Figure 6, we can see that the consecutive frame multi-scale ICP refinement clearly improves the camera pose trajectory accuracy compared to the trajectory generated by the maplab SLAM algorithm.

For all our dense 3D reconstruction, we run on a desktop computer with 128 GB system memory to enable reconstructing at the finest TSDF voxel resolution possible (*i.e.*, 2 - 5 mm depending on the 3D scene volume dimension).

4.3 Virtual Reality

One of the features of the handoff between the reconstruction and the VR headset that became valuable was the simple drag and drop into the file system. It meant that one person could take the output of a scanned site, zip the data, and send it to other people to test. The ease of sharing is a valuable feature described more in the discussion.

While the drag and drop import meant the user didn't have to process the files, it does have a few downsides. Firstly, when using an Android based headset like the Quest 2, plugging the headset into the computer doesn't initially expose the headset's filing system. The computer must be given access to the filing system first, which is a trivial process of clicking 'accept' on a prompt in the headset. The larger problem is that there's no indication to the user of where to drop the files in the headset. Because of permission settings in Android applications, each app only has access to specific folders. As a result, the files from the reconstruction must be placed in `/Internal shared storage/Android/data/com.DefaultCompany.MayaArcheologyVR/files/Sites/`.

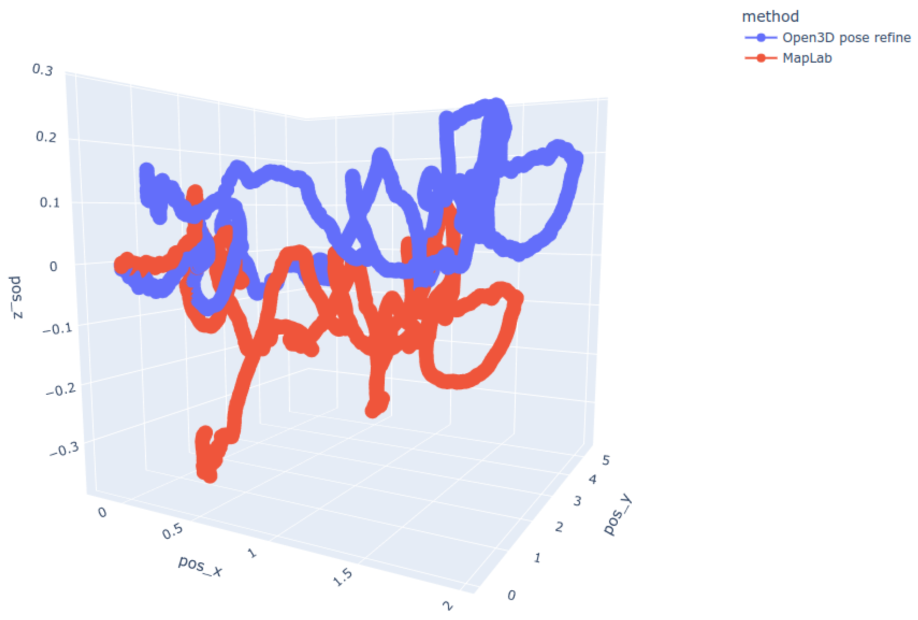
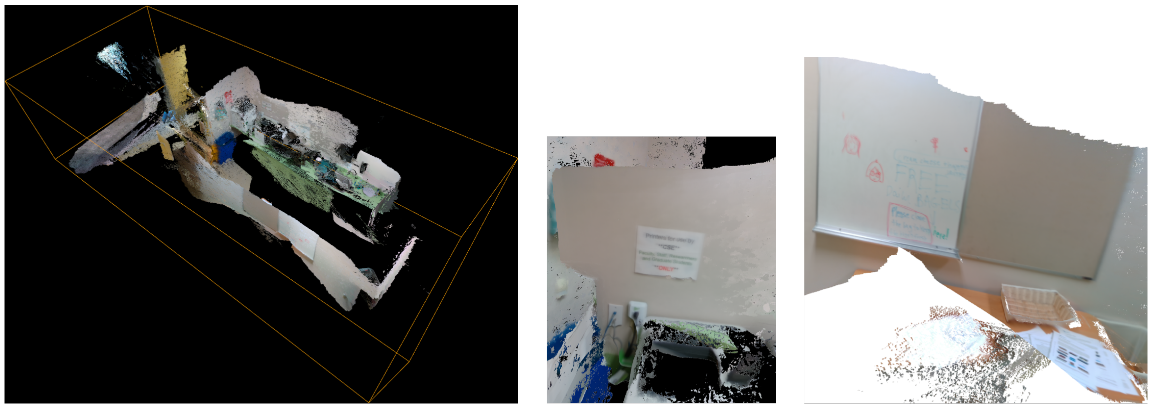


Fig. 6. Camera pose trajectory comparison between the maplab estimate and the Open3D refined trajectory. The ending position of maplab trajectory is quite far off from the starting position while they are pretty close to each other in the Open3D refined trajectory.



(a) A high level of detail can be quickly attained via the reconstruction

(b) Printed text is legible, though slightly blurry

(c) Content on a whiteboard is clearly visible in reconstruction

Fig. 7. Results of the offline reconstruction

Once in VR, the user appears in an empty space where they don't have any indication of what to do. Hitting the 'y' button opens up the main menu and provides indication of what all of their controls are. This controller help text is placed directly on the virtual controllers (rather than in the previous system's static images) allowing the user to easily form a mental model of the controls. The main menu also has a series of buttons that highlight as the user passes their controller cursor over them.

Once a site to load has been selected, the site begins loading. A progress bar appears and the menu is automatically closed once the site has finished loading. The user can enable the minimap and will see their location in the map as well as the location of tour points throughout the map. Using both the translation and teleportation movement controls are simple, natural, and standard across other VR applications.

In terms of presence[6], we consider that a cave environment should feel dark, even scary, should feel as if it constrains movement and visibility. In the VR cave environments, we notice a difference between the first and latter experiences. The first experience in one of the reconstructed caves captures these feelings quite well - the single light source causes a spookiness of not being able to see everything around you, and the spiders that crawl on the walls can be a bit frightening. These features increase the sense of presence the user feels when they first enter the virtual space.

However, as the user moves around and becomes more familiar with the environment, little things degrade at this sense of presence. First, the fear element dissolves when the user eventually realizes that their virtual body cannot be harmed[7] by the spiders and once they understand the simple algorithm behind their movement. Second, the incongruence between physical and virtual movement is very jarring. While the head collider does force the user into ducking while in the caves, and this does improve the sense of presence, the difference between when the physical body overlaps with the virtual cave walls but the virtual body collider does not (and vice versa) causes a huge reduction in the sense of presence. The body collider had to be made extremely thin so that the user could move through tight spaces (where they might squeeze sideways through a space). This means that when the user isn't turned sideways, their collider doesn't reflect their shoulder, arm, and hip width. Allowing their physical body to clip through the virtual walls. Additionally, the collider getting stuck on something that, if physically in the cave, would be easy to step over or around also detracts from the experience.

Lastly, the teleporting and translation capabilities detract from the sense of presence further. Having a large space to physically walk around in allows the user to role-play slightly. This small increase in physical activity lends itself to the believability of being there. These movements align the physical body's agency with the virtual space, whereas the passive controller based movements make the suspension of belief difficult as the user becomes aware of the discrepancy between their body and their visual experience.

As a whole, the sense of presence is a mixed response. When initially entering the site, the darkness of the environment has a very strong effect on the user. As the subtle misalignments between the physical and virtual space are experienced, the user, over time, is less and less convinced that they are in the space and more and more aware of the limitations of their virtuality.

5 DISCUSSION

By creating a system that allows for sites to be loaded at runtime, alongside the reduction of expertise needed for the scanning procedure, this platform takes a large step toward democratizing the digitization of cultural and ecological sites. This democratization is important because of the previously mentioned phenomenon of site selection favoring the preservation of Western sites [13]. By reducing barriers to entry, the system allows for a potentially larger number of sites to be digitized and shared around the world.

The downside of this, however, is that the digital replica can be used as justification for the destruction of these sites. It is not difficult to imagine land developers arguing that because a digital version of a physical location exists, the

[6]Presence is a common subjective description of how much the user felt like they were really in the virtual environment and is distinct from the quantitative measure of immersion [21]
[7]The feature of virtual harm is a result of the Proteus Effect [26]

physical location can be repurposed toward maximizing industrial or commercial utility. As our evaluation highlights, while the digital replica can be convincing, it does not create the same sense of presence as actually being there. For this reason, we are concerned that expanded digitization will lead to the loss of the subtler aspects of experiences of visiting these sites.

5.1 Limitations and Future Work

There are many expansions to the system we would like to see. In terms of reconstruction fidelity, the current system is capable of extreme detail, but we noted some dimensional inaccuracy with mesh components not quite aligning. Tracking could be improved with different cameras, and depth accuracy could also be improved via Lidar or Time of Flight cameras. The downside of this approach is that both of these cameras tend to be far more expensive than the low cost RealSense cameras we used. In the other direction, current phones and tablets are beginning to get various RGB-D cameras like the system on some iPhone and iPads. While less accurate than other systems, non-researchers are more likely to have easy access to these camera systems than they are to others. As this system matures, we would like to see multi-camera support so that users can use whatever cameras they have at their disposal. In addition, a system that combines the richness of dense SLAM with the modularity of sparse SLAM can be used which would improve tracking performance.

Additionally, while the hand-off between reconstructing models and displaying them in VR is relatively easy, it isn't as convenient as it could be. Given more time, we would have liked to setup an online database where models from the reconstruction would automatically be uploaded, and could be downloaded to a particular headset from inside the application, or even shared with others around the globe. A easier to reach goal would be add GUI support for automatically uploading the output of the reconstruction onto a plugged in headset.

Beyond this, there are always small optimizations that can be made both in the VR and reconstruction. The current VR system for loading in high level of detail meshes can be time consuming and could be optimized by saving the Unity optimized mesh data to the file system after loading. Additionally, recent advances by Unreal Engine's Nanite system allow extremely high triangle counts on low power systems like those powering Android VR systems. Migrating to Unreal Engine would allow for the removal of the balance between fidelity and VR performance.

In terms of the reconstruction, improvements are constantly being made in the fidelity of reconstructions. While we tested and reviewed several algorithms an their performance, recent AI-based systems like those from Nvidia [19], so phenomenal results, including the ability to reconstruct vegetation from images. While this technology has a long way to go, it is promising and will likely soon overtake methods like those we used.

5.1.1 Expandability. The tools and packages we used for this system allow for it to be easily expanded to support more platforms. While this wasn't a goal of the project, it was a consequence of the software we used. Namely, a cross-platform VR game engine, a cross-platform Scanning GUI, and a container for the scanning and reconstruction algorithms. Since the realtime SLAM and offline reconstruction run in a docker container, we added docker and our particular docker image installation scripts to the GUI. This allows the user to not have to figure out how to install dependencies or the reconstruction software, as it is all automated. Because both docker and the GUI run cross-platform, it also means that in order to expand the system to support more platforms, new installation scripts for those platforms need to be added and little more.

For the VR system, while it has the potential for expansion to more platforms, there are a few more hurdles to get past first. Being built on Unity's game engine, the system can build to target both Android and PC VR systems as well as WebVR and even AR devices. And while the system currently runs in both PC VR based Oculus Rift and Android VR based Oculus Quest platforms, expanding to other PC VR systems requires the input system to be updated to support those systems (currently, the input system only supports Oculus systems). This is a matter of importing the Mixed-Reality Toolkit [8] and switching input handling in the `InputManager.cs` script to use the new package.

6 MILESTONES

The milestones that were established at the start of the quarter were :

(1) Literature review on promising SLAM and reconstruction algorithms
(2) Mirroring laptop display onto a tablet for visualization
(3) Port existing VR application to Quest 2
(4) Runtime import of 3D models into VR applications
(5) Data capture with realsense into rosbags
(6) Install build and test SLAM algorithms on standard datasets
(7) Hardcode scenes to models
(8) Update VR interface to populate sites from filesystems
(9) Dense 3D reconstruction and export as 3D model
(10) SLAM on collected data
(11) Break 3D models into smaller models
(12) loading multiple models into VR
(13) End-to-end integration <— MVP
(14) Exporting mesh with config file
(15) Recording Audio Tours
(16) Test multiple cameras
(17) Optimize reconstruction for hard light

The MVP of the project was set at the end-to-end integration step. The MVP describes an automated pipeline that allows a user to scan a site using the Realsense camera, with real-time visual feedback while scanning, and see it in VR without having to manually run different commands.

We achieved our MVP by week 8, where we were able to record sequences within the CSE department using the realsense camera, densely reconstruct the CSE kitchen and corridor and load it into VR, with administrative features for manipulating with the scene model. We also went beyond the MVP and performed mesh export using config file, and recording audio tours.

However, towards the end of Week 7, we realized that to make this pipeline more usable, a GUI was necessary that can enable the user to go through with this pipeline, by just using a few clicks and inputs. Therefore, we slightly shifted our focus and spent a considerable amount of time developing the GUI to interface with the different components of the project. The GUI was largely a success as it achieved the overall project purpose of creating a single lightweight tool that can walk a user through all the steps without any difficulty. Thus, the final updated milestones completed were

(1) Creating docker containers for each step, and creating shell scripts to launch and run the containers.
(2) Developing a GUI that demonstrated the purpose and overall goal of the project.

[8]https://docs.microsoft.com/en-us/windows/mixed-reality/develop/install-the-tools?tabs=unity

As a result, not enough time was dedicated towards improving the implementation by using multiple cameras and optimizing for hard light. These two were nonetheless set out as reach goals at the start of the project.

Due to the extensive ground work done in the first couple of weeks both in terms of literature overview, system design choices and setting the goals for the project, we did not face any major problems during the course. We were able to solve most of the minor problems and obstacles that we faced; a few of them include

- *Messy VR code* : The existing VR code was not well documented, and took a considerable amount of time to get familiar with
- *Importing model color data* : Another significant issue that we faced was in exporting and importing 3D models while preserving color information. After the 3D reconstruction and mesh splitting, most softwares/libraries did not provide compatible formats for preserving and interpolating color to the point cloud while importing into the VR application. After some research into the matter, a library that supported this (PyVista) was found and the code was re-written with this library
- *Making code expandable* : Since SLAM and reconstruction methods are constantly improving, constant decisions had to be taken to ensure that the system could be expanded later, and different components/modules could be replaced with equivalent ones. To support expandability, we made use of Docker, Electron GUI, and SteamVR with Android which are all largely cross platform.

7 CONCLUSION

In this paper, we presented a pipeline to carry an unskilled user through the process of scanning, reconstruction, setting up tours, and showcasing digital reconstructions of physical environments. Through the automatic installation and simplistic interface, this system has the potential to democratize the digitization of the physical world, sharing and preserving sites of important cultural heritage both for those who may never see the sites otherwise, and for those looking to study and conduct research on these sites. In the end, we hope this work can be used as both springboard and reference for a future where preservation can be conducted by anyone with a camera and isn't isolated to a small population of experts; helping to remove what biases and blind spots we, as researchers, have from the digitization process.

REFERENCES

[1] Alehegn Adane, Assefa Chekole, and Getachew Gedamu. 2019. Cultural Heritage Digitization: Challenges and Opportunities. *International Journal of Computer Applications* 178 (July 2019), 1–5. https://doi.org/10.5120/ijca2019919180

[2] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. 2011. Building rome in a day. *Commun. ACM* 54, 10 (2011), 105–112.

[3] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. 2020. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. *CoRR* abs/2007.11898 (2020). https://arxiv.org/abs/2007.11898 arXiv: 2007.11898.

[4] F. D'Agnano, C. Balletti, F. Guerra, and P. Vernier. 2015. TOOTEKO: A CASE STUDY OF AUGMENTED REALITY FOR AN ACCESSIBLE CULTURAL HERITAGE. DIGITIZATION, 3D PRINTING AND SENSORS FOR AN AUDIO-TACTILE EXPERIENCE. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-5/W4 (Feb. 2015), 207–213. https://doi.org/10.5194/isprsarchives-XL-5-W4-207-2015

[5] Jakob Engel, Vladlen Koltun, and Daniel Cremers. 2017. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence* 40, 3 (2017), 611–625.

[6] Jakob Engel, Thomas Schöps, and Daniel Cremers. 2014. LSD-SLAM: Large-scale direct monocular SLAM. In *European conference on computer vision*. Springer, 834–849.

[7] Quentin Kevin Gautier, Thomas G. Garrison, Ferrill Rushton, Nicholas Bouck, Eric Lo, Peter Tueller, Curt Schurgers, and Ryan Kastner. 2020. Low-cost 3D scanning systems for cultural heritage documentation. *Journal of Cultural Heritage Management and Sustainable Development* 10, 4 (Jan. 2020), 437–455. https://doi.org/10.1108/JCHMSD-03-2020-0032 Publisher: Emerald Publishing Limited.

[8] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang, and Guoquan Huang. 2020. Openvins: A research platform for visual-inertial estimation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 4666–4672.

[9] Gabriele Guidi, Michele Russo, Sebastiano Ercoli, Fabio Remondino, Alessandro Rizzi, and Fabio Menna. 2009. A multi-resolution methodology for the 3D modeling of large and complex archeological areas. *International Journal of Architectural Computing* 7, 1 (2009), 39–55.

[10] Matthew Klingensmith, Ivan Dryanovski, S. Srinivasa, and Jizhong Xiao. 2015. Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially Hashed Signed Distance Fields. In *Robotics: Science and Systems*. https://doi.org/10.15607/RSS.2015.XI.040

[11] Blase LaSala. 2019. *Creating a High-Fidelity Interactive Simulation of the Timpanogos Cave System from a Terabyte-Scale Terrestrial LiDAR Dataset.* Master's thesis. The University of Arizona, United States – Arizona. https://www.proquest.com/docview/2354751770/abstract/602645964B034162PQ/1 ISBN: 9781392496367 Publication Title: ProQuest Dissertations and Theses.

[12] William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics* 21, 4 (Aug. 1987), 163–169. https://doi.org/10.1145/37402.37422

[13] Zinaida Manžuch. 2017. Ethical Issues In Digitization Of Cultural Heritage. *Journal of Contemporary Archival Studies* 4, 2 (Dec. 2017). https://elischolar.library.yale.edu/jcas/vol4/iss2/4

[14] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. 2015. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE transactions on robotics* 31, 5 (2015), 1147–1163.

[15] Raul Mur-Artal and Juan D Tardós. 2017. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics* 33, 5 (2017), 1255–1262.

[16] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. KinectFusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. 127–136. https://doi.org/10.1109/ISMAR.2011.6092378

[17] George Pavlidis, Anestis Koutsoudis, Fotis Arnaoutoglou, Vassilios Tsioukas, and Christodoulos Chamzas. 2007. Methods for 3D digitization of Cultural Heritage - ScienceDirect. *Journal of Cultural Heritage* 8 (2007), 93–98. https://doi.org/10.1016/j.culher.2006.10.007

[18] Massimiliano Pieraccini, Gabriele Guidi, and Carlo Atzeni. 2001. 3D digitizing of cultural heritage. *Journal of Cultural Heritage* 2, 1 (Jan. 2001), 63–70. https://doi.org/10.1016/S1296-2074(01)01108-6

[19] Soumik Rakshit. [n.d.]. Extracting Triangular 3D Models, Materials, and Lighting from Images. https://wandb.ai/geekyrakshit/Extracting%20Triangular%203D%20Models/reports/Extracting-Triangular-3D-Models-Materials-and-Lighting-From-Images--VmlldzoxOTQ2MDEy

[20] Thomas Schneider, Marcin Dymczyk, Marius Fehr, Kevin Egger, Simon Lynen, Igor Gilitschenski, and Roland Siegwart. 2017. maplab: An Open Framework for Research in Visual-inertial Mapping and Localization. *CoRR* abs/1711.10250 (2017). http://arxiv.org/abs/1711.10250 arXiv: 1711.10250.

[21] Thomas Schubert, Frank Friedmann, and Holger Regenbrecht. 1999. Embodied Presence in Virtual Environments. In *Visual Representations and Interpretations*, Ray Paton and Irene Neilson (Eds.). Springer, London, 269–278. https://doi.org/10.1007/978-1-4471-0563-3_30

[22] Dinar Sharafutdinov, Mark Griguletskii, Pavel Kopanev, Mikhail Kurenkov, Gonzalo Ferrer, Aleksey Burkov, Aleksei Gonnochenko, and Dzmitry Tsetserukou. 2021. *Comparison of modern open-source visual SLAM approaches*. Technical Report arXiv:2108.01654. arXiv. http://arxiv.org/abs/2108.01654 arXiv:2108.01654 [cs] type: article.

[23] C. Bane Sullivan and Alexander A. Kaszynski. 2019. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software* 4, 37 (May 2019), 1450. https://doi.org/10.21105/joss.01450

[24] Vladyslav Usenko, Nikolaus Demmel, David Schubert, Jörg Stückler, and Daniel Cremers. 2019. Visual-inertial mapping with non-linear factor recovery. *IEEE Robotics and Automation Letters* 5, 2 (2019), 422–429.

[25] Thomas Whelan, Stefan Leutenegger, Renato Salas-Moreno, Ben Glocker, and Andrew Davison. 2015. ElasticFusion: Dense SLAM without a pose graph. Robotics: Science and Systems.

[26] Nick Yee, Jeremy N. Bailenson, and Nicolas Ducheneaut. 2009. The Proteus Effect: Implications of Transformed Digital Self-Representation on Online and Offline Behavior. *Communication Research* 36, 2 (April 2009), 285–312. https://doi.org/10.1177/0093650208330254 Publisher: SAGE Publications Inc.

[27] Guoxiang Zhang, YangQuan Chen, and Holley Moyes. 2018. Optimal 3D Reconstruction of Caves Using Small Unmanned Aerial Systems and RGB-D Cameras. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. 410–415. https://doi.org/10.1109/ICUAS.2018.8453277 ISSN: 2575-7296.

[28] Qian-Yi Zhou and Vladlen Koltun. 2013. Dense scene reconstruction with points of interest. *ACM Transactions on Graphics* 32, 4 (July 2013), 112:1–112:8. https://doi.org/10.1145/2461912.2461919

[29] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2018. Open3D: A Modern Library for 3D Data Processing. *CoRR* abs/1801.09847 (2018). http://arxiv.org/abs/1801.09847 arXiv: 1801.09847.