Maya Archaeology : Pipelines and toolkit to reconstruct historical sites in virtual space

Ruixiang Qi
Department of Computer Science
University of California San Diego
rqi@ucsd.edu

Giovanni Vindiala
Department of Computer Science
University of California San Diego
gvindiol@ucsd.edu

Joseph Warmus
Department of Computer Science
University of California San Diego
jwarmus@ucsd.edu

Haoqi Wu
Department of Computer Science
University of California San Diego
haw175@ucsd.edu

Abstract

All locations of historical significance are at risk of being vandalized, but if they existed in an approximated digital world they could never truly be forgotten or destroyed. In order to mitigate this possible loss of experience we provide a method of preserving these environments digitally by using Remote scanning, 3D Procedural Software, Game engines, and VR technologies. We are able to reconstruct and build immersive interactions to be used in cultural heritage preservation, archaeological research, and education. We've assembled a 3D Automation Toolkit that focuses on transforming point cloud data into optimized 3D Assets using Houdini, and created an Unreal Engine Model Viewer to simulate lightning and render out images for archaeological publications. These assets are also used as environmental art in our virtual 3D interactable tour system created using the Unity game engine. Together these different systems provide a platform for newcomers to quickly become citizen scientists that can contribute to virtually preserving cultural heritage in an ever changing environment.

1 Introduction

Digital Documentation of excavation M7-1 at El Zotz, Guatemala has been done over several field seasons using Terrestrial LiDAR. In 2014 the Faro Focus 3D 120 was used to generate a precise 3D model to aid Archaeologist in tunnel mapping. In 2016, structure M7-1 was excavated even further, and a new Point Cloud was acquired to merge with previous data and so on. Additionally, during these field expeditions Structure from Motion (SfM) was performed to obtain color information from key areas and create physically-based material (PBR) samples of the different environmental components. The reconstructed geometry generated straight from these point clouds are not ideal for many common mediums such as an app, website, or game console. Thus an optimization procedure is required to reduce the complexity of these models and make them more compatible with consumer electronics. This work has traditionally been done by skilled artists that must manually retopologize, or "simplify" the complexity of the vertices and edges comprising the 3D geometry among other tasks. We pursued a method that would allow us to feed in a series of images, dense point clouds or meshes, and then transform this data into game-ready assets ready for archaeological visualizations.

1.1 Data Acquisition through Remote Scanning

1.1.1 Terrestrial Light Detection and Ranging (LiDAR)

The process for Terrestrial LiDAR Scanning (TLS) generally involves setting the desired scan density, which in affect increases the number of points collected and time needed per scan. In the case of the Lieca BLK360 TLS, medium quality was sufficient for our purposes and took 3 minutes per scan. Each scan occurred more than 1 meter apart depending on if all the environment's geometry could be reached from the previous scans location. The set of individual point clouds obtained was then registered using visual alignment in Lieca's propriety software Cyclone Register 360. The output of all this is several standard file format .ptx files all in alignment with each other, which can then be brought to external software for further 3D point cloud and mesh processing.

1.1.2 Structure from Motion (SfM)

SfM point cloud derived models are able to store accurate RGB data at a higher resolution, but with lower geometric accuracy. Since RGB information collected during SfM have greater accuracy, they can be useful in projecting color information back onto the Mesh generated by LIDAR. We can do Fine registration with the Iterative Closet Point (ICP) algorithm on the 2 Models to share the same coordinate space. Performing SfM on larger environments can be time consuming in collecting and processing data. In addition to this are the extra constraints of consistency in scene lighting, and obtaining enough coverage and overlap between photos of the entire scene. Thus we conceptually decompose an environment into it's most abundant components and perform SfM on these components under controlled lighting conditions for the purpose of high-detail surface scanning applicable toward environmental texturing. The m7-1 excavation was classified into 2 materials, it's general structure component of limestone stucco and the natural elements comprising the walls and ceiling of the excavation. This surface information can then be input into newer high-level development tools that utilize machine learning to upscale and synthesis material variations.

1.1.3 Aerial Light Detection and Ranging (LiDAR)

The Foundation for Maya Cultural and Natural Heritage (PACUNAM) funded a LiDAR initiative that surveyed a portion of the Guatemalan jungle including the El Zotz region of Protected Biotope San Miguel La Palotada. The LiDAR rapidly emits pulses of light that transmit through multiple surface intersections. Each intersection returns a signal that captures it's Time-of-Flight and from that a 3-dimensional coordinate and classification can be assigned to each point. The ground points can be extracted to reveal the terrain which can then be used to generate an accurate landscape than can serve as a canvas to align TLS and SfM scans to capture a broader environment.

2 Technical Material

In this section, we describes the technical details of each of the three components of our project, which are the automation toolkit, model viewer and VR application.

2.1 Automation Toolkit

The automation toolkit takes high-polygon rough photogrammetry as input and automatically performs clean-up, mesh-reducing, UV-unwrapping, and baking to produce a low-poly game-ready output. In this section, we discusses the tools we used and specific steps we took to create our automation toolkit.

2.1.1 Houdini

The automation toolkit involves many different complex operations, including auto UV unwrapping and auto poly-reducing. Most existing 3D image processing software do not have all the functionalities we need. After a lot of research and many experiments, we found Houdini to be the suitable software that can help us automate the whole process. Specifically, Houdini is a 3D procedural software in which users can design a workflow to automate different processes. For example, in our project, as illustrated in figure 1, we created a workflow in Houdini and made a series of nodes to automate different tasks. Specifically, the workflow contains a workflow for pre-processing (import-file node, clean-up node and define-up-vector node), a workflow for mesh-reducing and auto-UV unwrapping, a workflow for assigning color ID and a workflow for baking. To creating such workflow, we learned from various literature and tutorials. Particularly, we learned a lot from Travis Hove's previous effort of producing an Automation Pipeline and reproduced a part of his work.[1]

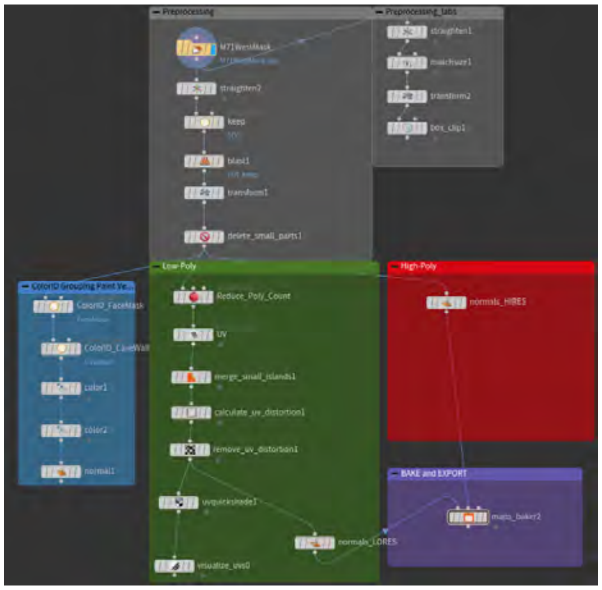

Figure 1: Houdini Automation Toolkit Workflow

The main reason Houdini contains all types of functionalities is that Houdini supports external third-party tools and packages. For example, although Houdini does not have an auto-UV-unwrapping tool itself, there is an extension called "SideFX labs" that provides the auto-UV-unwrapping functionality we need.

2.1.2 Preprocessing

In preprocessing, our main task is to remove unnecessary parts in the original mesh to produce a clean mesh that only contains parts that we want. For example, the image on the left of figure 2 shows an example of the original imported mesh that contains extra structure that we do not want, and the image on the right of figure 2 shows the output of the preprocessing step, in which extraneous parts are removed and up-vectors are defined.

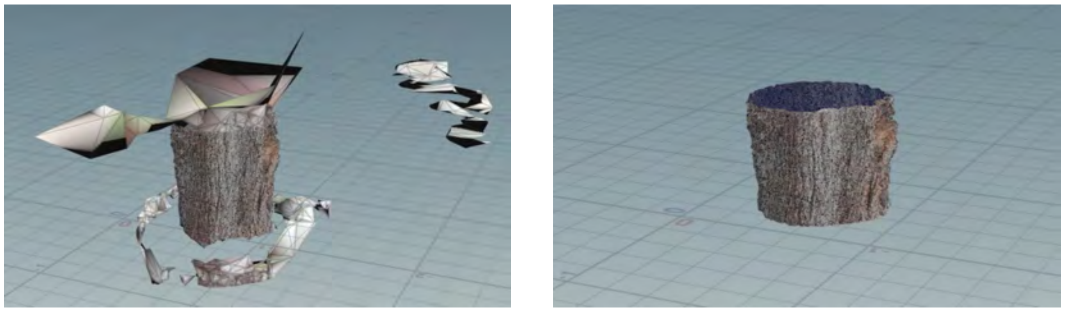

Figure 2: Raw imported mesh vs. Mesh after preprocessing

The workflow for pre-processing is illustrated in figure 3.

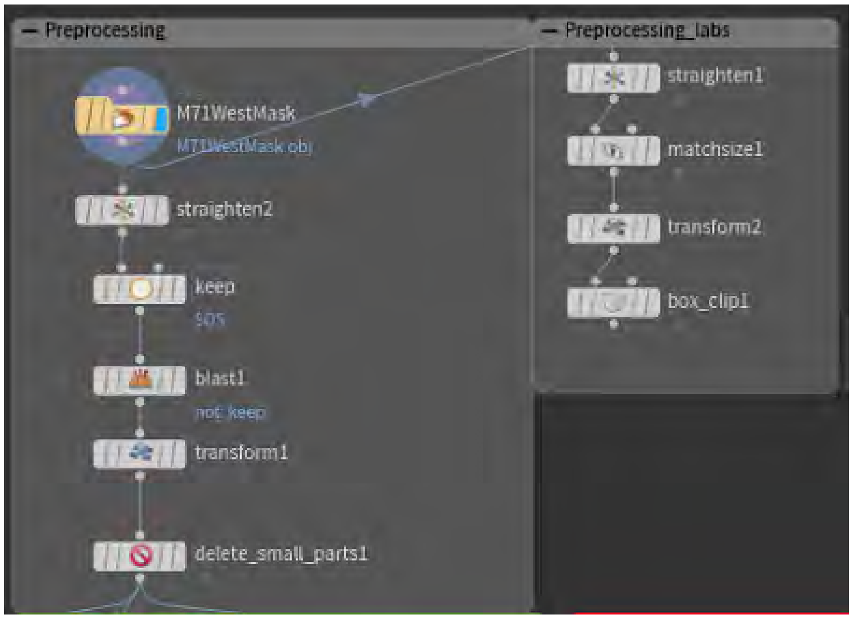

Figure 3: Preprocessing Workflow

The specific procedures and nodes we used as listed as followings:

1. We import a High-poly Mesh which is a high-poly .obj file that is produced by photogrammetry using a "File" node
2. We use three nodes to remove the unnecessary part in the original raw mesh and clean it up. Specifically, we first use a "Labs_Straighten" node to define an up-vector so that the mesh is facing the correct direction (this is actually an optional step). Then we use the "Group_Create" node to specify which part of the original input is redundant and use the "blast" node to remove the redundant parts. Specifically, we used a "square box" in the "Group_Create" node to specify which parts should not be removed because we had a square-shaped input mesh
3. With a "transform" node , we are able to make our mesh stay at the origin of the houdini

2.1.3 Low-poly & High-poly processing

In low-poly processing, our task is to reduce mesh's poly and make its size suitable for a game asset. This step is one of the most important procedures. Figure 4 shows the major output during low-poly processing, including the output for poly-reducing, UV-unwrapping

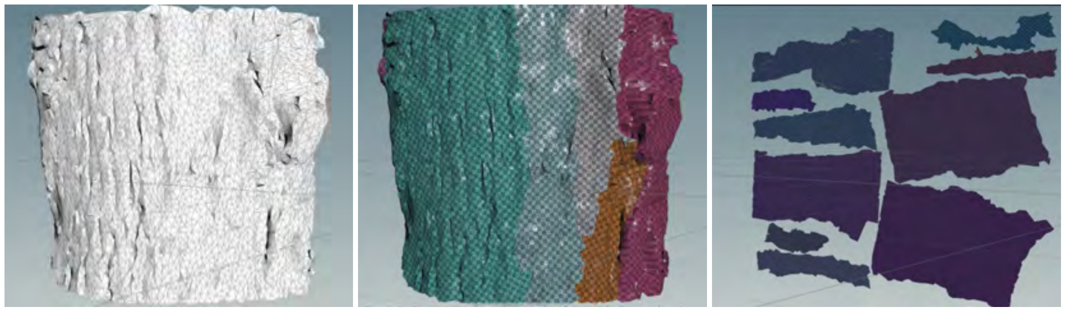

Figure 4: Major outputs in low-poly processing (output for poly-reducing and UV-unwrapping)

The workflow for low-poly processing is illustrated in figure 5.

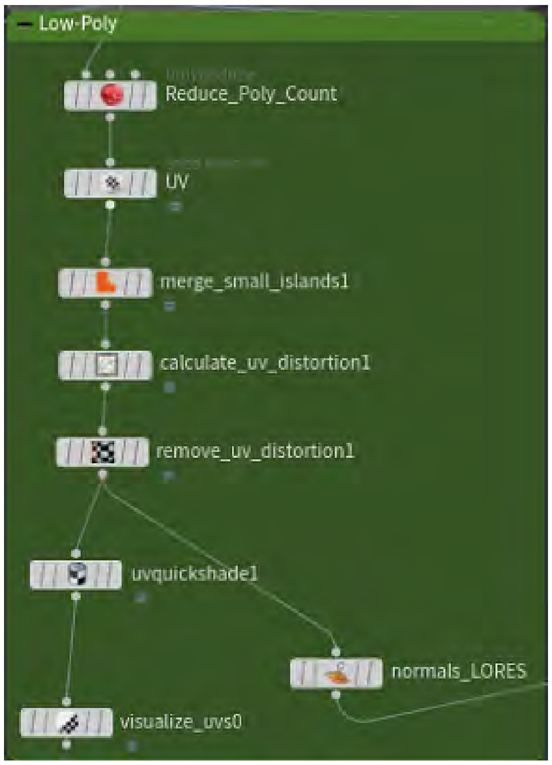

Figure 5: Low-poly processing Workflow

The specific procedures and nodes we used as listed as followings:

1. We use a "PolyReduce" node to automatically reduce the poly count of our input.
2. We use a "Labs Auto UV"(UV) node to automatically UV unwrap our mesh. This node is from the extension "sideFXLabs".
3. We first use a "Labs merge small islands" ("merge_small_islands1") node to merge the small islands in the mesh to further reduce the polygons. Then we use "Labs Calculate UV Distortion" node to find UV distortions and then connect it to a "Labs Remove UV Distortion"node to remove the UV distortions.
4. We use the "UV Quick Shade" node to shade our unwrapped UV and use a "Labs UV Visualize" to visualize the unwrapped UV.
5. As the final step to prepare for the baking, we use a "Normal" node to compute normals.

As for high-poly processing, We actually do not need to make additional steps for high-poly processing. We simply use a "Normal" node to compute normals to prepare for baking to Low-poly. (As illustrated in Figure 6)

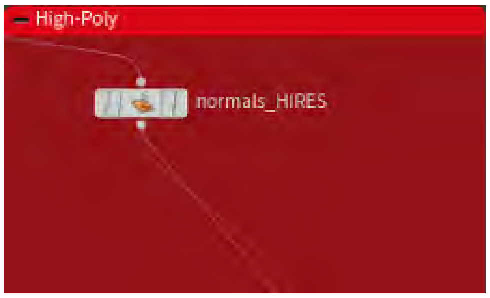

Figure 6: High-poly processing Workflow

2.1.4 Baking and Exporting

Finally, we bake the details in high-poly mesh onto the small, polished, and UV-unwrapping low-poly mesh to create our final output. Figure 7 shows an example of our final output.

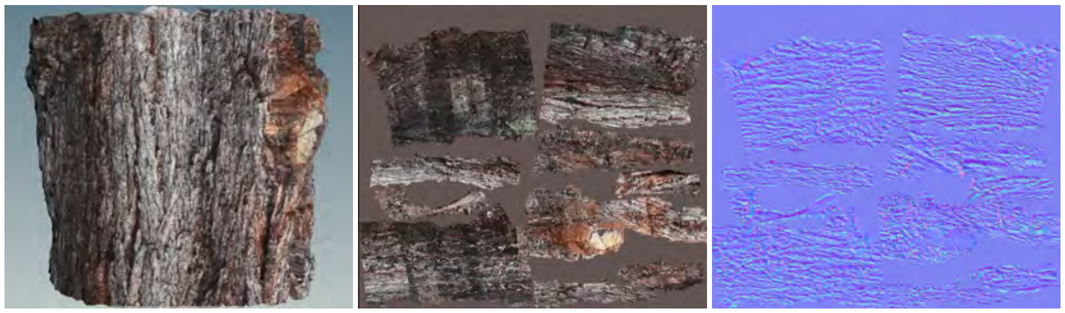

Figure 7: Final Output

The workflow for baking and exporting is illustrated in Figure 8.

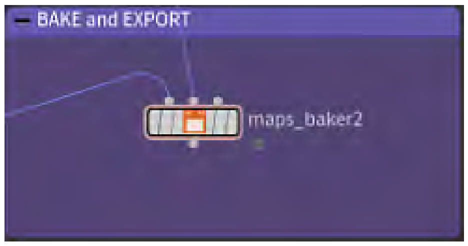

Figure 8: Baking and Exporting Workflow

The specific procedures and nodes we used as listed as followings:

1. We use a "Labs Maps Baker" node to bake high poly onto low poly.

2. we use an "FBX output" node to export the result. (this feature is not available in a free version of Houdini)

2.1.5 Color ID Grouping

Although we eventually did not finish auto texture rendering in substance automation toolkit, we did manage to assign different textures different color IDs, which is an important step towards automatic texture rendering. As illustrated in figure 9, we are able to assign different colors to different parts that should be rendered different textures.

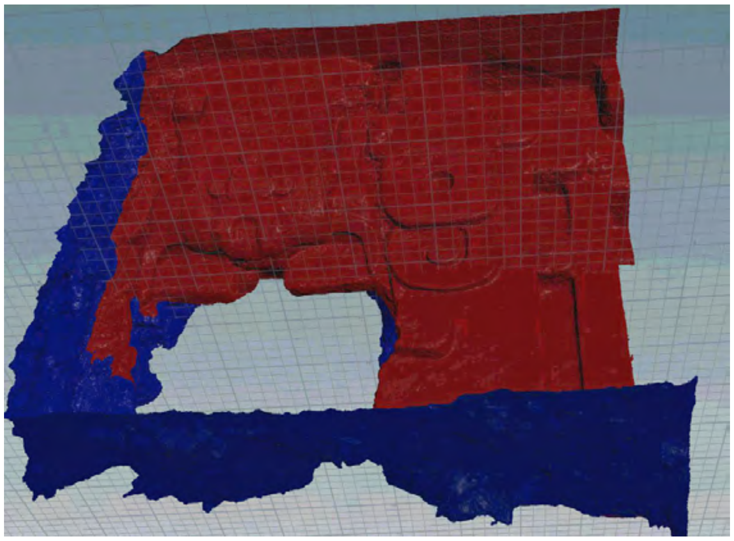

Figure 9: Example of color id grouping

The workflow for colorID grouping is illustrated in Figure 10.

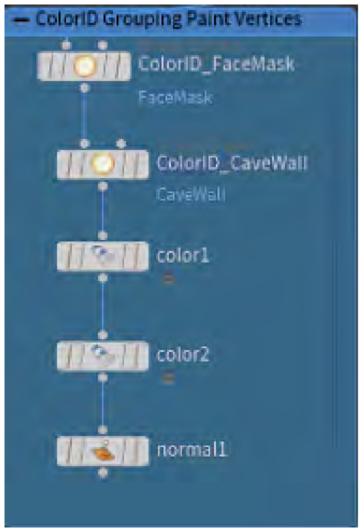

Figure 10: ColorID Grouping Workflow

The specific procedures and nodes we used as listed as followings:

1. We use "Group_Create" nodes to specify different parts that should be rendered different textures(in our case, "cave wall" and "face mask")
2. We use "Color"("color1","color2") nodes to render different colors to different output from the previous two "Group_Create" nodes.

2.2 Model Viewer

2.2.1 Engine Details

The Model Viewer application is developed in Unreal Engine 4, version 4.25. The application was developed on Windows 10. Non-default Unreal plugins used are Color Wheel version 2020.1.4, Easy

File Dialog version 2.0, RuntimeMeshImportExport version 1.11, and RuntimeTransformer version 1.0. The application uses Unreal Engine 4's ray-tracing features to produce high quality images

2.2.2 Runtime Model Loading Backend

Model loading is done through use of the RuntimeMeshImportExport Unreal Engine Plugin, which enables models to be loaded at runtime from the user's PC. The plugin allows for files of a variety of formats, and theoretically has texture support, though the files we tested on didn't appear to work. We ended up using a workaround for texture loading, detailed below. Point cloud loading and display is also supported through the LiDAR Point Cloud Plugin included with Unreal.

- **Texture Loading** - Because of the difficulty with getting correctly formatted files to be loaded by the RuntimeMeshImportExport Plugin, we instead opted for a practical solution of expecting the textures to be located in the same folder as the model file with a particular suffix. Unreal's built in texture loading functionality is used to load these textures and apply them to the loaded model's dynamic material. Specifically, the model viewer can load in a diffuse, specular, and normal texture along with the model. The suffixes are "-DIFFUSE", "-SPECULAR" and "-NORMAL" respectively.

- **File Dialog** - File dialogs are used to make locating the files to be loaded much simpler. This is accomplished in a very straightforward manner using the File Dialog plugin, which allows for string filepaths to easy be obtained for use by the RuntimeMeshImportExport plugin. These filepaths must be converted to absolute filepaths, or the application can fail to properly load files in build versions due to issues with relative filepaths, which are Unreal's default.

2.2.3 Camera Control

The camera control utilizes Unreal's built in camera options. A spring arm component is used to achieve the standard camera behavior, of rotation around a center point. Additionally, free camera and perspective toggles were added to make the camera usage experience more intuitive. The free camera simple uncouples the camera from the spring arm component and allows the user to move around the scene freely with WASD and the mouse. Perspective/orthographic toggle is done by editing the camera settings. Unreal's cinecamera (cinematic camera) was used over a standard camera, as it provides extra options in terms of aspect ratio, depth of field, and other cinematic effects, as it is intended for cutscenes rather than gameplay.

2.2.4 Lights

The three types of lights in use by the Model Viewer are point, directional, and HDRI. HDRI is the most generally useful, as it provides a good looking background and realistic looking ambient light. It uses a cubemap to define the environment, and is implemented through Unreal's built in HDRIbackdrop. We included 4 cubemaps that can be switched between through the GUI. The point and directional lights are handled through spawning a point or directional light actor into the environment, and can be moved through the movement gizmo, detailed below.

- **Movement Gizmo** - To allow for lights and the model in the scene to be manipulated, we used the RuntimeTransformer plugin, which provides editor-style movement gizmos to be used to manipulate objects in the scene, in our case lights and loaded models. The blueprint allows for objects to be selected via clicking and dragged. Through an additional GUI along the bottom-right edge of the screen, the user gains access to translation, rotation, and scale, as well as grid snapping settings. The gizmo itself, which appears when selecting an object by clicking on it, allows for free drag movement, as well as axis constrained movement by clicking on the desired axis gizmo and dragging it.

- **Color Wheel** - We use the Color Wheel plugin to allow the color of the lights to be changed dynamically. The color wheel menu appears when a light is selected, and allows for light color to be selected from a radial color menu, and intensity to be selected via a slider.

2.2.5 User Interface

The User Interface is implemented through use of Unreal's UI Widget system. There are buttons and dropdowns, which allow for various functions to be activated and selections to be made. There are buttons which spawn lights, trigger the model loading dialog, trigger the render dialog, toggle the HDRI light, remove the currently selected light/model, and quit the application. There are also dropdowns for HDRI light selection and render resolution selection.

- **Gizmo Interface** - The Gizmo interface is separate from the main user interface, and interacts only with the Gizmo controls. These are based heavily on the example blueprints provided by the RuntimeTransformer plugin. It provides rotation, translation, scale, and snapping settings.
- **Color Wheel** - The Color Wheel interface allows for light color to be selected as described above, and is based on the Color Wheel plugin's example blueprints.

2.2.6 Rendering

Output image rendering is done through Unreal's HighResShot console command, which allows for renders to be create at any resolution. The Model Viewer provides the user the option to select an output location through the Easy File Dialog plugin, and to select a multiplier for the output resolution. Though the process for obtaining a screenshot is theoretically simple, a lot of code is required to make sure the output image is presentable. After all, a screenshot will simply take a photo of everything on the screen. Particularly, all lights placeholder meshes, gizmos, and UI needs to be absent in the final image. To this end, all light meshes are made invisible during the render, and reenabled after, and the currently selected object is deselected. There are several alternative methods that we explored before arriving at HighResShot, that could work for other similar projects. The Movie Render Queue is a very high quality option, but is only available at runtime in Unreal 4.26+. Another option explored was NVIDIA Ansel, which is a very powerful tool but is unfortunately incompatible with Unreal Engine's raytracing and AMD hardware.

2.3 VR Application

The VR application is developed using Unity game engine (version 2020.3.3f1). The hardware used for development is Oculus Quest 2 with link cable.

2.3.1 Hand Tracking

To enable hand tracking, we used the OVR Camera Rig and OVR Hand in Oculus Integration package. The hand tracking can only be functional under the tracking space of OVR Camera Rig. We added the OVR Camera Rig to the existing XR Rig and disabled the camera in OVR Camera Rig so it does not conflict with the XR setup. We used the finger pinching functions from the OVR plugins, which map the pinching of left index finger to selection pointer, pinching of left middle finger to toggling between opening and closing menu, and pinching of right index finger to teleportation. One challenge we faced was that Unity only supports hand tracking feature in the editor and disables it when building the app. It is fixed by adding user permission for hand tracking through modification of the Android Manifest file.

2.3.2 Navigation System

The first part of our navigation system is the hologram minimap. For each site, we created a hologram minimap by using the site collision mesh and downscale it to a smaller size. We also created a transparent material with emissions to make it look like a hologram. When the hologram is turned on, there will be a red sphere indicating the current position of the player. For each frame, the script accesses the local position of user's transform with regard to the world coordinates, and updates local position of the player indicator on the hologram. Since player indicator is a child of the hologram, its position will reflect the user's position in the site.

The second part of the system is the waypoint feature. To set up waypoint feature, we created a yellow sphere as the waypoint indicator on the hologram. The waypoint indicator has a collider and can be selected by the selection pointer on user's hands. User can move the waypoint to any position on the hologram, and the position of the actual waypoint is updated in the site using similar logic as the player indicator. We also display an arrow in front of the camera, which points to the position of the actual waypoint. Its forward direction is updated every frame and always points to the waypoint.

2.3.3 Creature System

We create a Creature Manager class to handle the instantiations and behaviors of the creatures. The first type of creature we added to the site is spider. We use the asset Animated Spider from the asset store for spider model and animations. For spider AI, we wanted to make them climb over the meshes of the site when the user is close. We initially tested the Unity built-in navigation system, but due to the complexity of the site meshes, we were unable to bake the navigation mesh. After experimenting different approaches, we used ray-casting to help the spiders navigate on the mesh of the site. We shoot a ray from the camera's position to a point in front of the spider (a small distance in spider's forward direction), and check the intersection of the ray with the mesh collider of the site models. With the intersection point, we move the spider to the position of intersection, and rotate the spider by the normal direction of the intersection point. Now the spiders can move on the walls and meshes with any shape. We perform distance checks in Creature Manager, and let the spider move when the user gets close.

We also added howler monkeys to the M7-1 site. We found online models for howler monkey and rigged the model by ourselves. We then produced the animations in each frame and attached the howling sound to the monkeys. The animated howler monkeys are placed on top of the tree outside the M7-1 site and continue howling when the player is within a distance.

2.3.4 Site Optimization and Modifications

In order to optimize the performance of our application, we perform several techniques for site optimization. First, we separated the sites into different scenes. Before scene separation, all the site assets are inside one main scene. The Site Manager class handles the functionality of enabling and disabling the site assets. However, having too many objects in one scene can lower the performance of the application even if some objects are disabled. Moreover, it is difficult to customize the sites separately when everything is together. Therefore, we remade the starting scene and separated the sites into individual scenes, and reworked the Site Manager class to handle scene loading. Currently the scenes are entirely separated and the events are handled in each scene individually. In the future, we plan to make more connections among the scenes by sharing the prefabs and variables.

Moreover, we used pre-baked light maps instead of the real-time lighting inside the scenes. The pre-baked light maps uses less computational resources than the real-time lighting, while maintaining the realistic lighting effect as the sites are static. To further improve the performance, we separated the site meshed into multiple small portions, and turned on occlusion culling to hide the meshes that are not seen by the user. However, we faced lighting issues when the occlusion culling is on, which was caused by a bug of Unity. We were unable to fix the issue and had to turn off the occlusion culling. In the future, we can implement manual culling features which hide the unseen meshes based on the position of the user.

We also merged the inside and outside environments of the M7-1 site. The meshes of the cave are merged together with the mesh of the outside hill. Now the user can directly travel from the hill to the cave just like at the real site.

2.3.5 Texture Switching

We manually created a different texture for the Diablo site. For each new texture, we created a separate material with the albedo and normal map. We placed a sphere next to the part of the site that has new textures, and attach a script to handle texture switching when the user touches the sphere. We tried to implement the dissolve effect for the switching process and we found tutorials for that using Unity shader graph. However, the usage of shader graph requires our project to use a different rendering pipeline, and we were unable to change the pipeline at this point.

3 Milestones

In this section, we listed our planned milestones at the beginning of the quarter and discussed the revisions that we made in the middle of the quarter. We also shared obstacles that we encountered when completing each milestones and how we overcome those obstacles. For the milestones that we did not complete, we discussed the main reasons that prevented us from accomplishing those goals.

3.1 Automation Toolkit

- Set up Houdini(Week 4 Ruixiang)
    - Completed. After deciding to use Houdini as our tool for automation toolkit, we read and watched tutorials and successfully downloaded and tested Houdini.
- Configure an AliceVision node (Week 5 Ruixiang)
- Incomplete. At first, we found that AliceVision node was unavailable in Houdini's free version. To solve the problem, we received help and got access to a paid version. However, even so, we still found many errors preventing us from properly using the node. We tried to seek help from online resources, but did not received enough instructions online that could help us identify the issues and fix the bugs. Luckily, the AliceVision node was not essential to the automation toolkit. We eventually used another software to produce the photogrammetry that we need and used the resulting photogrammetry as our input for the automation toolkit.
- Configure a clean up node (Week 5 Ruixiang)
    - Completed. We first learned from the experience report of Travis Hove and reproduced a part of his work.[1] Then, we adjusted the workflow and added additional nodes the make the cleaning-up processing work better for our project.
- Configure a mesh reduction node (Week 6 Ruixiang)
    - Completed. Using Houdini's "poly-reduce" node, we were able to reduce our high-poly mesh to low-poly. We were also able to specify how much polygons we want to get rid of. Fortunately, we did not meet a challenge to configure this node.
- Configure a UV Unwrapper node (Week 7 Ruixiang)
    - Completed. Although Houdini does not have an auto UV-unwrapper itself, we were able to find an extension called "sideFX labs" that supports auto-UV unwrapping. Additionally, except for configure a UV Unwrapper node, we also added many additional nodes for fine-tuning the output and displaying the UV unwrapped output.
- Configure a baking maps node (Week 7 Ruixiang)
    - Completed. At first, our output has darker color than we expected, but we finally managed to solve this issue. Additionally, we were not able to output our result at the beginning since Houdini's free version does not support "FBX" format output. This problem was also eventually solved after we gained access to a Houdini paid version.
- Configure a ColorID node (Week 7 Giovanni)
    - Completed. By combining "Group_Create" nodes and "Color" nodes together, we were able to assign different colors to different parts that should be rendered different textures.
- Configure a Substance Automation kit (Week 8 Ruixiang, Giovanni)
    - Incomplete. We successfully set up the python environment for Substance Automation kit. We also managed to use python scripts to assign some example textures to our mesh. However, we were unsuccessful of assigning out own textures to our mesh. The reason is that our texture does not have the same format as textures in Substance's tutorials. We tried to seek help from substance in their slack channel, but did not receive enough help and instructions from them about how to overcome such difficulties. Given the steep learning curve of substance automation kit, we did not have enough time to learn to adjust our python scripts to make it work for our own textures.

3.2 Model Viewer

- Enable camera control (fps and trackball) around model, and orthographic/projection view switching (Week 5, Joseph)
  - Completed. Orthographic and projection view switching was relatively trivial, but camera control was a somewhat more complex endeavor, as it had to be very intuitive and straightforward. This milestone took more time than an it might have for an experienced Unreal Development as Joseph had to get over a lot of Unreal's initial learning curve for this step.
- Enable run-time model and texture loading (Week 6, Joseph)
  - Completed. This step was more challenging that it initially seemed. The plugin used for this step, RuntimeMeshImportExport, is pretty complex and not terribly transparent. Debugging this plugin was time consuming, especially when it came to texture loading. The plugin is supposedly compatible with built-in textures in .FBX files, but we could not get it to work, so a workaround with separate texture files had to be implemented, which is detailed in our documentation on our website.
- Enable point cloud loading/rendering (Week 6, Giovanni)
  - Completed. This step was straightforward, as Unreal has a built in LiDAR Point Cloud Plugin perfectly suited to this purpose.
- Adding a Graphical User Interface (Week 6, Joseph, Giovanni)
  - Completed. GUI's are easy to create in Unreal Engine 4, and a simple but practical GUI for our application was created. It gives the user access to scene manipulation, mesh import with file browsing, and rendering control.
- Allow lights to be added and manipulated, allow mesh material to be edited (Week 7, Joseph)
  - Mostly completed. Directional and point light addition and manipulation was completed through the Runtime Transformer plugin. The Runtime Transformer plugin is quite complex, and took a bit of time to properly implement. Mesh material editing was left on the cutting room floor as we decided to have the material's properties be entirely determined by the loaded textures, making getting good looking results simpler for the end user. Future iterations of the application might look to add a separate material editing sub-menu, however, or provide some presets.
- Enable high quality, path traced renders to be produced at run-time using the Movie Render Queue (Week 8, Joseph)
  - Completed with compromise. The inital plan was to have path-traced renders be produced by the user with Unreal Engine 4.26+'s runtime Movie Render Queue functionality. This would allow for very high quality, photorealistic renders to be created. However, we realized about halfway through development that the RuntimeMeshImportExport plugin is only compatible with Unreal Engine 4.21-4.25, making having both runtime mesh import and path-traced renders not possible. As runtime mesh import was an essential facet of the project, we decided to nix path traced rendering. In its stead, we were able to implement super-resolution ray-traced renders through the High Resolution Screenshot functionality of Unreal Engine 4. This solution is lower quality than Movie Render Queue's, and has the restriction that aspect ratio must be the same as the Unreal Engine window, but still is able to produce high quality images that are sufficient for the original purpose of the application.
- Enable image-based lighting and background manipulation (Week 8+, stretch goal, Joseph)
  - Completed. Using Unreal's HDRI backdrop features, several options for simple image-based lighting backgrounds were implemented. This feature makes setting up a good looking shot very quickly easy. The main challenge with this step was finding high-quality cubemaps to include with the Model Viewer. One backdrop was one that was included with Unreal Engine, and the others came from a pack purchased online with a free-use license.
- Enable distances between points to be measured and displayed (Week 8+, stretch goal, Joseph)

  - Not completed. The decision was made early on to normalize the size of models being loaded into the model viewer application to prevent a model from being way too large, small, or off center upon loading, disorienting the user and causing additional setup time to be required. Unfortunately, this operation, which is completed by the RuntimeMeshImportExport plugin, does not preserve the original size information of the mesh loaded in. This would make this function impossible to implement without requiring additional metadata, and was left on the cutting room floor due to its relative unimportance.

- Integrate geospatial features via Cesium (Week 8+, stretch goal, Joseph)
  - Not completed. The actual features we would implement were never decided on, and it was decided that the time spent on Cesium integration would be better used to polish core features and squash bugs, rather than add more features for the sake of it.
- Enable culling volumes to be used (Week 8+, stretch goal, Joseph)
  - Not completed. Unreal Engine is a game engine, and not a 3D modeling application. Because of this, to the best of our knowledge, the sort of feature where an individual mesh is manipulated at runtime, is very difficult to do or impossible. Given that a culling volume is setting the visibility of only part of a mesh, this goal was found to be not possible, at least with our knowledge of Unreal Engine.
- Allow mesh normals to be inverted for an "X-Ray" style internal view (Week 8+, stretch goal, Joseph)
  - Not completed. Similar to the previous milestone, this feature doesn't seem to be possible within Unreal Engine 4 at runtime. Inversion of mesh normals can also be done relatively easily inside of 3D applications like Blender or Maya, so this feature was not completed.
- Allow models to be viewed in VR (Week 8+, stretch goal, Joseph)
  - Not completed. This feature was left on the cutting room floor due to its difficulty relative to its usefulness. Similar to the Cesium integration, this feature was content that wasn't directly useful to the core functionality of the application, creating a render of a model. In the end, it was not found to be worth the substantial effort to integrate VR while there was still work to be done on polishing and debugging the core functionality, especially given the VR functionality's dubious usefulness and the fact that we have a separate VR application already.

3.3 VR Application

- Display hand tracking models in application (week 4, Haoqi)
  - Completed. This step was relatively difficult in the early stage of the project, when we were still trying to get familiar with the whole project. After we managed to understand the structure and functionalities of the existing application, we added this feature by importing the Oculus Integration package and modify the player asset.
- Map hand gestures to control system (week 5, Haoqi)
  - Completed. For this milestone, we learned about the API for hand tracking and connect them to the existing features like selection and teleportation. The major challenge we faced was that Unity does not support hand tracking in the built app. We managed to find resources with tutorials on how to modify the Android Manifest file to enable this feature.
- Create hologram minimap for navigation (week 6, Haoqi)
  - Completed. This step was easier than expected, since the mapping of the uesr's position to the hologram was relatively easy with local coordinate system. Most efforts were spent merging the new feature with the existing menu and control systems, as well as creating site-specific holograms.
- Implement waypoint feature for navigation (week 7, Haoqi)
  - Completed. Similar to the hologram minimap, the mapping of positions is relatively easy with local coordinate system. We spent more efforts on developing the user

interactions (select and move the waypoint on hologram), which requires techniques such as ray-casting and collision detection.

- Merging M7-1 Interior and Exterior as larger environment. (Week 7, Giovanni)
  - Completed. This step was straightforward, as we had already made the inside and outside environments to working assets.
- Add creatures to the sites (week 8, Haoqi, Giovanni)
  - Completed. We added spiders and howler monkeys to the site. This milestone was more complicated than we expected. The navigation of spiders, in particular, could not be implemented with normal methods. After experimenting with different approaches, we came up with the ray-tracing technique to navigate the spiders. The process of rigging the model for howler monkeys and creating animations was also time-consuming, but we were satisfied with the results we achieved in the end.
- Implement texture switching for the sites (week 9, Haoqi, Giovanni)
  - Completed without special effects. Creating a new texture and apply it to the current site was relatively easy. Nevertheless, we spend a good amount of time and efforts trying to create dissolve effects for texture switching using the shader graphs, yet we were unable to make it work.
- Site modifications and VR experience optimizations (week 9, Haoqi, Giovanni)
  - Completed. This milestone was relatively difficult. Every modification we made involved reworking a portion of the existing project, and we were very careful during the process to not break the project. The occlusion culling took much of our efforts but we were unable to resolve the lighting issue caused by it. But with the experience we can create manual culling in the future.

4 Conclusion

The technical workflow for the creation of 3D assets for digital media as traditionally done through game development is being abstracted away. We are moving toward a path that will automatically transform scans into photo-realistic reconstructions. The days of an archaeologist manually surveying and creating illustrations to primarily record findings are over. Remote sensing is king. If our aim is for precision then we can use Terrestrial and Aerial LiDAR to map out excavations and surrounding regions. We can quickly collect a series of images using an SLR or Smartphone to record a spatial snapshot of an environmental feature. We can use real-time Simultaneous Localization and Mapping (SLAM) to quickly record a tunnel system. The Open Geospatial Consortium (OGS), and Environmental Systems Research Institute (ESRI) are embracing the integration of ArcGIS and Cesium ecosystems into game engines to unlock features that would take years of Research and Design internally. We are now able to assemble highly accurate representations of geographic regions using remote sensing and game development technologies.

References

[1] Travis Hove. *Game ready asset from photogrammetry*. sidefx, Mar 2020.