Dive Reconstruction Pipeline Using Underwater Film

Michael Nixon
Fulton School of Engineering
Arizona State University
Tempe, Arizona
mnixon6@asu.edu

***Abstract*–This document discusses the process of developing a workflow for creating 3D models from video footage using ffmpeg and the python API for Agisoft Metashape. Additionally we discuss the results achieved by various inputs into the pipeline and the ease of use for users with no knowledge of the underlying systems.**

I. INTRODUCTION

We use a selection of footage[1] to attempt to generate 3d models from the Agisoft Metashape algorithm, leveraged through the provided Python API. We begin by preprocessing the footage into a series of images either through the Metashape user interface (UI) or ffmpeg, a free file conversion software.

II. PROJECT GOALS

Originally, the goal of this project was to recreate a dive site and use this model in Gazebo Classic to perform ORB SLAM using the pre-built underwater drone. After many technical difficulties using the outdated Gazebo Classic, the project was revised to look at the python API, and image processing methods to create a pipeline for scene reconstruction using dive footage.

Thus, this project aims to create an easy to follow and repeatable workflow that produces reasonable 3d models of provided video without intimate knowledge of the programs at work.

III. METHODOLOGY

In this section we discuss the methodology used for creating the pipeline. We begin with using ffmpeg to convert any footage we wish to use into a series of images that Agisoft Metashape can consume to use in the process of generating the 3D models. We then cover the various use cases of Metashape.

A. Using ffmpeg

Following the directions for installing ffmpeg [2] we gain the ability to process video footage, converting from video to a series of photos using the command:

```
ffmpeg -i input.mp4 -vf fps=1
output%d.png
```

This outputs a series of images into the current directory named output1.png, output2.png, etc. based on the length of the footage and the selected framerate. These files get output into the directory in which the script is running.

B. Agisoft Metashape Video Import

Agisoft Metashape comes with built-in functionality to import video and convert into a series of images similar to what we accomplish with ffmpeg. The import allows selection of a custom frame rate at which to create the series, or three given presets: Small, Medium, and Large time steps. From this import, Agisoft creates the images from the video and adds them to the chunk as if a series of images had been imported. From here, we return to the standard workflow covered later.

C. Agisoft Metashape Python API

We use the Python API for Agisoft Metashape [3] to align the input photos. This process begins by loading all of the photos output from ffmpeg. To do this we use Metashape's addChunk() command to create a new chunk, and then the addPhotos() method with the input being the image list to add all of the photos to the new chunk.

In the case that a multispectral camera is being used, the rgb, nir, and possible other formats of each image are imported as separate lists and shuffled together as tuples and added to the chunk using the same addPhotos() command, but with the inputs as the image list, and Metashape.MultiplaneLayout.

We now match all of the frames given in the chunk using the chunk.frame.matchPhotos() function, giving the downscale factor as input. Finally, we use the cunk's alignCameras() function to align the views that have been imported.

With the frames now matched, we continue on to use the buildDepthMaps() function to create the depth maps for the chunk, before finally calling the buildModel() function to generate the output we desire.

In the project we have included a python file with some basic code chunks that lay out the foundation for the pipeline, but the code we used to generate the models has not been included due to it including keys to use the professional license of Agisoft Metashape, and very detailed records of the host computer's file structure.

D. Limitations of the Python API

While the API is rather thorough and well documented, there is no mention of the video import function that exists within the main Agisoft Metashape UI. This means that rather than being able to import many of the standard video formats, users must preprocess their video in order to generate the photo series that then gets input to the python API.

E. Using the Agisoft Metashape UI

The UI includes a workflow tab that streamlines the 3d model creation. When the user first opens the menu, the only available option is to import images, similar to the addPhotos() command from the API. After this step is completed, the menu makes the option to align photos available. Next, the generate model option is selectable, with the final option we need being the generate textures functionality. Each of these steps allow for different settings to be selected.

When aligning photos the user can select to preorder the photos sequentially or allow Metashape to attempt matching in any order. The model can be generated from a variety of settings, but due to the results achieved we will not discuss the implications of these options. Similarly, the textures have a variety of settings to select which we will not discuss.

IV. RESULTS

In this section we discuss the results that were achieved using each of the photo series generated from the gathered footage. The information covered is: image processing presets, alignment percentage, number of tie points, number of points in the point cloud, and 3D model faces.

A. Main Wreck

From Agisoft, the best model possible from the footage was generated using an fps of one. From the one fps photo series, the internal camera alignment algorithm was able to align 46/66 of the images provided (69.7%). This is likely due to the camera drastically changing orientation between frames due to the recorder dropping the camera. While agisoft can recover the camera poses when the sample rate is higher for the photo series, the slower pace of the more clear sections of footage causes issues with the photo alignment, as agisoft metashape struggles to differentiate between frames within the given noise caused by being underwater.

We also choose sequential photo preselection, which does not affect the results, but given the photo series is sequential in nature, this reduces processing time. We also allow the algorithm to run on the highest selectable settings unless otherwise specified

In generating the model, the internal algorithm uses 23,016 tie points, and generates a point cloud of 330 thousand points. The generated model consisted of 502 thousand faces.

Processing this footage generates the most recognizable model, with the clearest correlation between the footage and model of all of the tested underwater footage, at all time steps.

B. Wreck 2

The footage of the second wreck was taken in a format that is processable by the video import functionality of the Agisoft Metashape UI, and as such was tested using various time steps to find the appropriate step for maximum model fidelity.

We keep sequential photo preselection, and allow the algorithm to run on the highest selectable settings unless otherwise specified.

Using the Medium Time Step preset, the footage generates 125 images, of which 72 are aligned (57.6%). This is likely due to the aforementioned pace issues found with underwater footage, as the beginning of the footage does not move much between polled frames. This chunk generates 74 thousand tie points, and generates a model with 25 thousand faces.

The issue that arises with the medium time step data is that while the model is detailed, it does not resemble the scene that is shown in the footage.

Using the Large Time Step preset, the footage generates 59 images, of which 59 are aligned (100%). This time step allows for more movement between polled frames, and allows agisoft to make clear distinctions between features within frames. This chunk generates 81 thousand tie points, and generates a model with 49 thousand faces.

This model resembles the original footage much better than the Medium Time Step model, but retains artifacts that do not make much sense in the greater context of the scene.

Additionally, due to the large lack of color variance in the second wreck's footage, the model returns mostly a teal-blue colored object that does not necessarily make it easy to differentiate which parts of the model correspond to the scene.

C. SDSU Aztecs Cup

To address some of the concerns that arose from the processing of underwater footage, we add another segment of footage taken panning around a glass cup to see what kind of results are possible from clear, no distortion footage. Additionally, a glass cup was selected to observe how the reflections of a shiny surface coupled with a transparent material is handled by the Agisoft Metashape reconstruction algorithms.

From the footage, which was processable through the video import feature, was tested using both the medium and small time step presets. The model generated by the small time step series is higher fidelity than the medium time step model.

The small time step model consists of 419 frames, of which 419 are aligned (100%). We hypothesize that the clarity of the footage allows for more precise feature detection across frames. The alignment generates 272 thousand tie points, a point cloud of 7.24 million points, and a 3d model of 503 thousand faces.

The model looks almost identical to the original cup, although the majority of the smooth glass is omitted. Only the base of the glass, the lip of the glass, and the engraved lettering plus logo are captured.

Additionally, as a monocular camera with no depth capture prevents agisoft from determining scale, which in this case causes the glass to be reconstructed at seven meters tall despite being the size of a standard 12oz glass.

V. COMPARISON

The model of the glass generated with the footage taken above water unsurprisingly generates a far superior model to the footage recorded underwater. Agisoft is able to align significantly more frames and process a much higher frame-per-second series.

Without technical knowledge of the Metashape model generation algorithm, we speculate as to the reasons for this difference, which is likely due to the clarity of the photo series allowing for much more precise feature matching. The inherent visual noise of being underwater creates too many uncertainties in the feature matching, and prevents the algorithm from matching higher framerate series.

VI. EVALUATION OF SUCCESS

We shall now evaluate the aforementioned measure of success: to create an easy to follow and repeatable workflow that produces reasonable 3d models of provided video without intimate knowledge of the programs at work. We find ffmpeg to be reasonably easy to install on a Windows machine, only requiring one line of terminal code, and another one to preprocess footage. From this, assuming the workstation already has Python3 installed, the python script runs flawlessly and generates the desired model with textures given the presets chosen in the script. Thus we have created an easy to follow workflow that creates repeatable results. The results, as discussed above, are variable based on the footage recording location and the pace at which the recording moves around the object we wish to recreate. However, the workflow can be followed generally without much knowledge of the underlying systems.

As such, we meet three out of four of the criterion we set out to achieve with the fourth being

somewhat covered based on the quality of the footage and some general finagling of the process inputs. The author views this as a moderate success, although a bit disappointing as a user will have to learn some of the python API in order to understand which inputs to the API are necessary to alter to achieve better results. Furthermore, sometimes trial and error is necessary to determine if the image processing step should become more or less fine-grained.

VII. CONCLUSION

The desired pipeline that we set out to craft, not only now exists, but also works decently well without alteration. While admittedly the results achieved given the presets in the code are not always the maximum possible fidelity for any given footage, we believe that the results are decent given no knowledge of the program is necessary.

The project was a good learning experience to better understand the workings of such a program, and the complexities of video processing. Learning to handle translucent objects and objects underwater was a quality experience.

ACKNOWLEDGMENTS

The author would like to thank the Fulton School of Engineering for their resources provided for the completion of this project. The author also thanks professor Jnaneshwar Das for his teachings and helpful feedback during the duration of this project. Finally the author would like to thank Will Kenyon for providing the underwater video footage used in parts of this project.

REFERENCES

[1]https://drive.google.com/drive/folders/1m4FwGlwLNGruyRBoN9a7rrYgZmr-yj5N?usp=sharing
[2]https://ffmpeg.org/download.html
[3]https://www.agisoft.com/pdf/metashape_python_api_1_6_0.pdf