Photogrammetry Workflow on an HPC system using Singulartiy, Python, and Jupyter Notebooks

(0) Introduction, Background, and Overview

The work described here builds off the Jupyter notebooks, analysis, and workflow documentation developed by student teams and Maurice Manning.

The Jupyter notebooks described here help the user create a 3D model using the Photoscan photogrammetry software, leveraging the computing power of a high-performance compute cluster. It assumes that the user's photographs are stored in a folder on Box, and the results of each step are saved back to Box. These Jupyter notebooks should be considered as tests to develop recommendations and findings and not necessarily specific solutions that should be implemented as is. Rather, users should modify the notebooks for further testing and experimentation and as is suitable to their specific needs, workflows, and frameworks.

As I discussed with Chris H, as of 7/22/19 I have now worked completely through the **PhotoscanWorkflow_7-22-19.ipynb** (which is a modified version of the Jupyter notebook **PhotoscanWorkflow.ipynb** created by Maurice Manning) and **BoxAuthenticationBootstrap_7-22-19.ipynb** (original written by Maurice Manning) notebooks on the Savio cluster (which currently reside in my home directory on Savio at the following path: **/global/home/users/myashar/photogramm_workflow/brc-cyberinfrastructure/analysis-workflows/notebooks)** twice, and, after reviewing with Chris, all of the output results (Photoscan project file, texture file, mesh file, photomosaic, OBJ, and JPEG files, etc.) seem reasonable -- keeping in mind, however, that **I have not yet, as of this time, carried out any of the offline/manual steps with photoscan that are specified in the Jupyter notebook (e.g., image masking, dense cloud cleanup, mesh cleanup). More information and a tutorial on some of these offline/manual steps can be found in Kea Johnston's Photogrammetry Tutorial.** The names of the output files and directories are

helmet_model.mtl (September 2018)

helmet_model.obj (model file) (September 2018)

photoscandemo.oc3

projectname_7-22-19.files (Photoscan project directory / files)

projectname_7-22-19.files.zip

projectname_7-22-19.jpg (texture file)

projectname_7-22-19.mtl

projectname_7-22-19.obj (model file)

projectname_7-22-19.psx (Photoscan project file)

projectname_7-22-19.tiff (orthomosaic file)

projectname_9-18.jpg (texture file)

projectname_9-18.mtl

projectname_9-18.obj (model file)

projectname_9-18.tiff (orthomosaic file)

and you can find most or all of them in my shared Box folder at this URL: https://berkeley.box.com/s/7h9lm64uaseglyg14zn93tk6j6psc8vz as well as in my Savio scratch space at /global/scratch/users/myashar/projectname/

Note that the **helmet_model.obj** file (not located in the Savio scratch space) was generated separately from the rest of the files (which were generated via the Jupyter notebook). The

helmet_model.obj was generated 'manually' using the other output files via the export option with the photoscan software installed on my Mac.

I have also included in the Box folder at the links specified above the SLURM log files (with extensive debug output) that were generated during the different steps when jobs were run on the Savio cluster (i.e. align photos, build dense cloud, export dense cloud, build mesh, build texture, etc.):

slurm-pull_images_from_Box.txt: execute image loading output

slurm-build_dense_cloud.txt: execute dense cloud build output

slurm-export_dense_cloud.txt: export the points output (export dense cloud)

slurm-build_mesh.txt: execute the mesh build output

slurm-build_texture.txt: texture and orthomosaic build output

slurm-export_photomosaic_format_output.txt: export photomosaic format output

slurm-generate_model_output.txt: generate model output

Everything covered in this document is described in much more detail in the shared google doc here:
https://docs.google.com/document/d/143zEgJhHhRgNspOB_hM23lc1x0OZK6qwb6VbmW5lytM/edit?usp=sharing

---

In summer 2018 Steve Masover and Quinn Dombrowski built / tested Singularity containers for Photoscan 1.4.3, using the latest release at that time of Singularity (2.6.0). They tested this on Savio to a limited extent, as described on Github. They put this Photoscan 1.4.3 Singularity container, **photoscan_1_4_3.simg,** in a shared/accessible location on Savio (**/global/scratch/groups/dh/)** where I was able to copy it to my scratch space on Savio.

In the process of working through and testing the Jupyter notebooks on the Savio cluster, I came across a few somewhat significant issues that would need to be addressed and which I suggested modifications for in some cases within the Jupyter notebooks. I described these in previous emails and you can also find them described and discussed within the extensive notes in the shared google doc HearstCAVE_Photogrammetry_Workflow. There are also a number of more minor 'idiosyncratic' issues that came up, i.e., issues or bugs that could come up for particular users (and that came up for me in some cases) depending on how things are set up for them on Savio, what their directory structure is like, their Savio environmental set-up, what software is installed or what modules are loaded,etc, but not something that would necessarily be generalizable to all users. These too are described in the shared google doc linked above. I summarize all of these issues, both significant and less significant, in the order in which I encountered them in the Jupyter notebooks as follows (and **I've also uploaded the modified Jupyter notebooks with some additional notes/comments and code boxes added in them in the shared Box folder here**). Some of the suggested changes in the Jupyter notebooks should be updated in the Github repo, whereas many others probably don't need to be:

(1) Project Setup:

(a) In order to access a Jupyter notebook on the Savio cluster, I follow the instructions in the UCB Research IT documentation for using Jupyter notebooks via the interactive Jupyterhub service here:
http://research-it.berkeley.edu/services/high-performance-computing/using-jupyter-notebooks-and-jupyterhub-savio#using-jupyter-notebooks-via-the-interactive-jupyterhub-service

Since we are testing and debugging the code here and not really running resource intensive computations, I spawn the “Local Server” job profile.

Through the Jupyter control page menu, I’m then able to navigate to:

/global/home/users/myashar/photogramm_workflow/brc-cyberinfrastructure/analysis-workflows/notebooks/PhotoscanWorkflow.ipynb

All of the Python code in PhotoscanWorkflow.ipynb was originally written by Maurice Manning.

Double click on the **PhotoscanWorkflow.ipynb** Jupyter notebook in order to access it.

(b) The user should make sure that they check that they are using Python version 3.5, and if not they should load the Python 3.5 module and/or even install Python 3.5 (e.g., with conda or pip) within their workspace on Savio and that it is activated. In my case and for debugging purposes, I added lines such as the following near the top of the first cell and in an added second cell of the Jupyter notebook:

```
# !conda create -n py3.5 python=3.5

# !conda install -n py3.5 pip

!module load python/3.5

# !conda list

!source activate py3.5

!python --version

!which python

!which python3

!python3 --version

!which jupyter

...

import sys

print(sys.executable)

print(sys.path)

print(sys.version)
```

just to make sure that the right version of python (3.5) (and Jupyter) was installed, loaded, activated, and utilized. So, for example, I obtain output that indicates that I'm currently using **Python 3.5.* :: Anaconda custom (64-bit)**

(c) In this part of the Jupyter notebook, the documentation there states that "***Before you get started, make sure you have a folder in Box for your project. That folder should have a folder inside it called "images" that has all the photos you want to use for your 3D model. This workflow will also create a new folder, "files", that contains the project files-- you don't have to set that up in advance.***" I did create the "images" folder inside the main project folder in Box (which for me was "myBoxFolderName") and I put the helmet images in the "images" subfolder, but, as far as I can tell, none of the code in the Jupyter notebook ever accesses or tries to download the images from this subdirectory on Box. Rather, the code looks for and downloads the images from the top directory on Box (e.g., "myBoxFolderName"). So, I needed to make sure that the image files were located in "myBoxFolderName" on box and not "images". The code would need to be modified to look for and download image files from the "images" subfolder on Box. Also, as far as I can tell, nowhere in the Jupyter notebook

does the workflow create a new folder called "files" containing the project files in Box. The code would need to be modified in the relevant place here as well.

There are 20 helmet images (JPEG files) with names such as “IMG_0336.JPG” and “IMG_0339.JPG”, etc. in the “myBoxFolderName” folder in my shared UCB Box folder. These files are also located in my scratch directory on Savio at /global/scratch/users/myashar/projectname/images/.

(d) Make sure that you're using the **latest version of the Chrome browser or Safari browser (for example)**, otherwise the widget text boxes and sliders might not work correctly or show up at all.

(2) Creating Project Folders

In the code box for creating the project folders, I added several print statements for debugging and sanity-checking purposes to check the paths to and names of all of the folders/directories and files that are to be created, e.g.,

```
print('home folder: ', homeFolder)

...

print('Singulariy Mount Folder: ', singularitymountfolder)

....

print('Box Project Folder: ',boxProjectFolder)

print('project file: ', projectFile)

print('data zip file: ', datazipFile)

print('data folder: ', dataFolder)

...

print('jpg file: ',jpgFile)

...

print('Slurm script: ', slurmScript)

print('exec script: ', execScript)
```

print('scratch exec script: ', scratchExecScript)

print('command script: ', commandScript)

The output that I get for all of the paths to and names of all of the folders/directories and files that are to be created is as follows:

user name: myashar

project name: projectname

run folder: /global/scratch/myashar/projectname/

home folder: /global/home/users/myashar/

Singulariy Mount Folder: /scratch/

Box Project Folder: myBoxFolderName

project file: /projectname.psx

data zip file: /projectname.files.zip

data folder: /projectname.files

jpg file: projectname.jpg

Scratch Image Data Directory: /global/scratch/myashar/projectname/images/

Slurm script: /global/scratch/myashar/projectname/slurmscript.sh

exec script: /global/scratch/myashar/projectname/execscript.sh

scratch exec script: /scratch/execscript.sh

command script: /global/scratch/myashar/projectname/commandscript.sh

Singularity Container Path /global/scratch/myashar/container/photoscan_1_4_3.simg

(3) Box Setup

(a) The documentation in the Jupyter notebook here states that "***if you run this notebook again for another project, you won't have to run BoxAuthenticationBootstrap.ipynb again, and can just continue to the code below***."

In my experience, this has not always been the case. In some cases even after successfully running everything in *BoxAuthenticationBootstrap.ipynb,* I've found that I still get error messages concerning expired tokens or invalid refresh token when running this part of **PhotoscanWorkflow.ipynb (Box Setup)** at a later time, such as the following,

BoxOAuthException:

Message: b'{"error":"invalid_grant","error_description":"Invalid refresh token"}'

Status: 400

URL: https://api.box.com/oauth2/token

Method: POST

and I then need to run *BoxAuthenticationBootstrap.ipynb* again.

(b) *BoxAuthenticationBootstrap.ipynb* is a utility script used to authenticate the user with CalNet and store the resulting auth and refresh tokens to be used in other scripts

When running *BoxAuthenticationBootstrap.ipynb,* you should check and make sure that boxsdk version 2.0.0a2 is installed and being utilized within your Savio workspace.

In my case, I added the line "!pip install --user boxsdk==2.0.0a2" in an earlier code box, as well as

```
import boxsdk

print(boxsdk.__version__)
```

in the first code box in *BoxAuthenticationBootstrap.ipynb.*

(c) In my case I also needed to replace

"**REDIRECT_URI = 'https://berkeley.account.box.com/login'**"

with

"**REDIRECT_URI ='https://berkeley.app.box.com/folder/0' "**

in *BoxAuthenticationBootstrap.ipynb,* but it's not clear to me as to whether this is necessarily generalizable or relevant to all other users.

During the authentication processes and steps here, along with the creation of initial tokens which are to be continually refreshed, and the creation of the globus client, make sure to choose the Standard OOAuth 2.0 (User Authentication) option in the Box Developers Configuration web page (at, e.g, https://berkeley.app.box.com/developers/console/app/842757/configuration). These steps

create a client to Box and verify that the authorization is valid. You can then verify the client by retrieving the name of the users root folder in Box, i.e.,

folder name: All Files

I believe that at this point the apptoken.cfg and app.cfg files in the (for my case) **/global/home/users/myashar/photogramm_workflow/brc-cyberinfrastructure/analysis-workflows/notebooks/** directory on Savio are created or updated.

(d) The user should also be aware of where the *.cfg files (e.g., **apptoken.cfg** and **app.cfg**) are actually being created in their Savio project space (e.g., **/global/home/users/myashar/photogramm_workflow/brc-cyberinfrastructure/analysis-workflows/notebooks/)** and where the code needs to access them. In some cases, one might need to manually copy them to their home directory if they aren't already there by adding a line into one of the code boxes here such as '!cp *.cfg ~/'

(e) For the section of the code where the Oath2 authorization is carried out, I sometimes received an error message (after running the relevant code box) with the final section of the error message being:

```
2

    3    """Callback for storing refresh tokens. (For now we ignore access tokens)."""

----> 4    with open(homeFolder,'apptoken.cfg', 'w') as f:

    5    f.write(refresh_token.strip())


TypeError: an integer is required (got type str)
```

(Sometimes, the previous 2 cells may need to be run twice in order to be able to move forward in running the subsequent cells in the Jupyter Notebook).

I found that I could safely ignore this message and move on to the next part of the Jupyter notebook.

(4) Create command scripts template

(a) For my case (which may not necessarily be generalizable to others) I needed to replace

execScriptTemplate = 'export RLM_LICENSE=5053@lmgr0.brc.berkeley.edu \n

/opt/photoscan-pro/photoscan.sh -r /scratch/commandscript.sh \n'

with

execScriptTemplate = 'export RLM_LICENSE=5053@lmgr0.brc.berkeley.edu \n

/usr/local/photoscan-pro/photoscan.sh -r /scratch/commandscript.sh \n'

in the section regarding the script for singularity to run. These commands create the execution shell script **/global/scratch/myashar/projectname/execscript.sh.** This execution shell script includes the license install as the first step each time to enable any HPC node.

(b) For issues that come up later, it may be useful for some users (though this may not be generalizable to all) -- depending on how their Savio workspace environment is set up and what the directory structure, etc. is like -- to add the following lines of code to the top of some of the code boxes here (particularly the one highlighted in yellow).

os.system('export XAUTHORITY=/global/home/users/myashar/.Xauthority')

os.system('export DISPLAY=:0')

os.system("module load qt")

os.environ['QT_QPA_PLATFORM']='offscreen'

(c) Also, it seems that the line for invoking the singularity command to run:

```
singularity exec --writable -B ' + runFolder + ':' + singularitymountfolder +
'   ' + singularityContainerPath + '   ' + scratchExecScript + '\n'
```

needs to be replaced by

singularity -d exec -B ' + runFolder + ':' + singularitymountfolder + ' ' + singularityContainerPath + ' ' + scratchExecScript + ' -platform offscreen \n'

so that it is compatible with the latest version of singularity that is being used here now. I use the '-d' flag here to obtain additional debug output information in the SLURM log files.

(d) Multiple times in the workflow, the project folder ("myBoxFolderName") needs to be updated on Box in order to enable making edits through the desktop version of Photoscan. The function for updating the project file implemented here sets up a utility function that pushes the project file ("projectname.psx") into the working directory on Savio. It assumes that the project folder will only contain one project file and one zipped archive.

(5) Fetching Images from Box

The code in this section retrieves the images from the project directory ("myBoxFolderName" in my case) on Box, and puts them in the project folder on the cluster that you specified above (**/global/scratch/myashar/projectname/images/**).

When you run it, it will check for the number of files in the folder on Box, and list each of them as it downloads. Please wait until you see *Download complete* before moving to the next step -- this can take some time if you have a lot of images, or large images.

In this section, it might be useful to un-comment / use the commented-out print statements for debugging purposes / sanity-checking.

Example output that I've obtained in this section:

boxProjectFolder: myBoxFolderName

retrieving images from folder: myBoxFolderName

downloading images to directory: /global/scratch/myashar/projectname/images/

number of files in top folder: 7

folder name: Astronomy-Microscopy Summit

folder name: My Box Notes

folder name: myBoxFolderName

```
targetfolderId:  53223729428
folder name:  myBoxFolderName_Test1
folder name:  savio_tests
folder name:  test_data
folder name:  workshop
number of files in target folder:  46
downloading:  IMG_0324.JPG
downloading:  IMG_0325.JPG
downloading:  IMG_0326.JPG
downloading:  IMG_0327.JPG
downloading:  IMG_0328.JPG
downloading:  IMG_0329.JPG
downloading:  IMG_0330.JPG
downloading:  IMG_0331.JPG
downloading:  IMG_0332.JPG
downloading:  IMG_0333.JPG
downloading:  IMG_0334.JPG
downloading:  IMG_0335.JPG
downloading:  IMG_0336.JPG
downloading:  IMG_0337.JPG
downloading:  IMG_0338.JPG
downloading:  IMG_0339.JPG
downloading:  IMG_0340.JPG
downloading:  IMG_0341.JPG
downloading:  IMG_0342.JPG
downloading:  IMG_0346.JPG
Download complete
```

Again, this output indicates and confirms that theimage files are being retrieved from the Box folder and downloaded to the project folder on the Savio cluster (e.g., **/global/scratch/myashar/projectname/images/)**

(6) Function for retrieving project file

The code in this subsection of the Jupyter notebook sets up a function to bring an updated project file (e.g., “projectname.psx”) back from Box, e.g. after you've made changes using the desktop version of Photoscan on a Mac OS or Windows system, and you want to run a subsequent processing step on the Savio cluster.

(7) Project Setup -- Set up job script: Run Job

The code in this section pulls the image files onto the Savio computing cluster and creates and uploads a Photoscan .psx file if it doesn't already exist.

(a) **Set up job script:** The code here sets up the job script that will be run on Savio, in order to pull in the *.jpg images from Box and, I believe, add them to the Singularity container, e.g.,

```
chunk.addPhotos( ['/scratch/images/IMG_0338.JPG', '/scratch/images/IMG_0325.JPG',
'/scratch/images/IMG_0333.JPG', '/scratch/images/IMG_0329.JPG',
'/scratch/images/IMG_0330.JPG', '/scratch/images/IMG_0334.JPG',
'/scratch/images/IMG_0337.JPG', '/scratch/images/IMG_0346.JPG',
'/scratch/images/IMG_0327.JPG', '/scratch/images/IMG_0336.JPG',
'/scratch/images/apptoken.cfg', '/scratch/images/IMG_0324.JPG',
'/scratch/images/IMG_0340.JPG', '/scratch/images/IMG_0342.JPG', '/scratch/images/app.cfg',
'/scratch/images/IMG_0339.JPG', '/scratch/images/IMG_0335.JPG',
'/scratch/images/IMG_0331.JPG', '/scratch/images/IMG_0328.JPG',
'/scratch/images/IMG_0326.JPG', '/scratch/images/IMG_0341.JPG',
'/scratch/images/IMG_0332.JPG'] )

doc.save(path="/scratch//projectname.psx", chunks = [doc.chunk])
```

(b) **Run job:** The code here runs the job on Savio, pulling in the images from Box, and will display the job ID.

When I first ran the job on the Savio Cluster I got the following error message:

ERROR : Could not open image /global/scratch/groups/dh/photoscan_1_4_3.simg: Permission denied

ABORT : Retrieval = 255

Which seemed to indicate that I needed to copy photoscan_1_4_3.simg to somewhere where I had the necessary read/write permissions.

So, **I copied photoscan_1_4_3.simg to /global/scratch/myashar/container/ and then re-ran everything**. Then got the following error

[myashar@jupyter projectname]$ more slurm-3289804.out

ERROR : Unable to open squashfs image in read-write mode: Read-only file system

ABORT : Retval = 255

as well as

QXcbConnection: Could not connect to display

Aborted

These problems were eventually solved by what was discussed in **#4(b) and #4(c) above**, and we get output such as the following as the job is running on Savio:

['Submitted batch job 4807120']

Execute image loading output: ['Submitted batch job 4807120']

4807120
JOBID PARTITION   NAME   USER ST   TIME NODES NODELIST(REASON)

4807120   savio projectn myashar PD   0:00   1 (Resources)

JobId=4807120 JobName=projectname

UserId=myashar(43974) GroupId=ucb(501) MCS_label=N/A

Priority=377307 Nice=0 Account=ac_scsguest QOS=savio_normal

JobState=PENDING Reason=Resources Dependency=(null)

Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0

RunTime=00:00:00 TimeLimit=00:02:00 TimeMin=N/A

SubmitTime=2019-07-18T16:51:29 EligibleTime=2019-07-18T16:51:29

StartTime=2019-07-18T16:53:07 EndTime=2019-07-18T16:55:07 Deadline=N/A

PreemptTime=None SuspendTime=None SecsPreSuspend=0

LastSchedEval=2019-07-18T16:53:04

Partition=savio AllocNode:Sid=jupyter:425871

ReqNodeList=(null) ExcNodeList=(null)

NodeList=(null) SchedNodeList=n0027.savio1

NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*

TRES=cpu=1,mem=62.50G,node=1

Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*

MinCPUsNode=1 MinMemoryNode=62.50G MinTmpDiskNode=0

Features=(null) DelayBoot=00:00:00

Gres=(null) Reservation=(null)

OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)

Command=/global/scratch/myashar/projectname/slurmscript.sh

WorkDir=/global/scratch/myashar/projectname

StdErr=/global/scratch/myashar/projectname/slurm-4807120.out

StdIn=/dev/null

StdOut=/global/scratch/myashar/projectname/slurm-4807120.out

Power=

Here, we see that the SLURM sbatch submission script and the Command script have been created or updated in the run/project directory on Savio (**/global/scratch/myashar/projectname)**, along with the SLURM output log file (**slurm-4807120.out).** The contents of the SLURM sbatch submission script (slurmscript.sh) are:

```
#!/bin/bash
# Job name:
#SBATCH --job-name=projectname
#
# Account:
#SBATCH --account=ac_scsguest
#
# Partition:
#SBATCH --partition=savio
#
# Wall clock limit:
#SBATCH --time=0:02:00
```

```
#
## Command(s) to run:
## source activate py3.5
singularity -d exec -B /global/scratch/myashar/projectname/:/scratch/
/global/scratch/myashar/container/photoscan_1_4_3.simg  /scratch/execscr
ipt.sh -platform offscreen
```

and the contents of the command script (commandscript.sh) are:

```
#!/usr/bin/env python3
import PhotoScan
import time
doc = PhotoScan.app.document
chunk = PhotoScan.app.document.addChunk()
chunk.addPhotos( ['/scratch/images/IMG_0338.JPG', '/scratch/images/IMG_0325.JPG',
'/scratch/images/IMG_0333.JPG', '/scratch/images/IMG_0329.JPG
', '/scratch/images/IMG_0330.JPG', '/scratch/images/IMG_0334.JPG', '/scratch/images/IMG_0337.JPG',
'/scratch/images/IMG_0346.JPG', '/scratch/im
ages/IMG_0327.JPG', '/scratch/images/IMG_0336.JPG', '/scratch/images/apptoken.cfg',
'/scratch/images/IMG_0324.JPG', '/scratch/images/IMG_0340.J
PG', '/scratch/images/IMG_0342.JPG', '/scratch/images/app.cfg', '/scratch/images/IMG_0339.JPG',
'/scratch/images/IMG_0335.JPG', '/scratch/image
s/IMG_0331.JPG', '/scratch/images/IMG_0328.JPG', '/scratch/images/IMG_0326.JPG',
'/scratch/images/IMG_0341.JPG', '/scratch/images/IMG_0332.JPG'
] )
doc.save(path="/scratch//projectname.psx", chunks = [doc.chunk])
```

(8) Upload Project File

(a) According to the documentation in the Jupyter notebook here, "***If there isn't a "data" subfolder inside your project folder on Box already, the code below will first create it...***"

I found that no such "data" sub-folder was created. In my case, the new Photoscan project file and all of the other data files were uploaded to my top level folder on Box, 'myBoxFolderName'. (In any case, I manually created a 'data' subfolder in the 'myBoxFolderName' folder on Box, which is currently empty).

This code box would need to be modified for this.

If a Photoscan project file with the same name already exists in the folder on Box, the code here will *not* overwrite it.

Note: New folders in Box can take several minutes to register, so this step may take a few minutes. If you get a "file not found" error in the next step, wait a minute or two and retry -- i.e., the box may need to be run twice for successful completion.

During this step, the project file "projectname.psx" (in my example) has been uploaded to Box.

(b) In some cases, I received the following warning/error message when running the code box here: (????)

I believe it can be safely ignored.

(9) OFFLINE: Image Masking

Please note that I did not carry out this step (perhaps further work to do in the future). It might be useful to think about how some of these offline (image processing/manipulation) steps can be done via python code within this Jupyter notebook rather than as an offline manual step using a desktop version of Photoscan -- or -- **if the desktop version of Photoscan could be launched from the BRC visualization node.** If that were the case, then there would be much less of a need to transfer files so frequently back and forth between Savio, Box and a Mac OS or Windows system where one would need to carry out the intermediate offline image processing steps with the desktop version of Photoscan. Rather, with the use of the desktop version of Photoscan run via the BRC visualization node, all or most of the photogrammetry workflow steps (within and outside of the Jupyter notebooks) could be carried out with Savio only with little or no need for the Box authentication steps and the use of another OS on another computer during the intermediate steps in the workflow. The fact that Jupyter notebooks can also be run on the BRC visualization node may also be advantageous for this proposed workflow.

(10) Align photos and Build Dense Cloud

(a) The code here retrieves the project file (e.g., projectname.psx), along with the data zip file (**projectname.files.zip)** from Box and downloads it to the run directory in the Savio scratch space (**/global/scratch/myashar/projectname/)**. This will ensure that we're working with the latest version, even if we've made changes offline (such as image masking).

The output from this step is as follows:

downloading: projectname.files.zip

data zip file download complete.

downloading: projectname.psx

new file: <_io.BufferedWriter name='**/global/scratch/myashar/projectname/projectname.psx**'>

project file download complete.

(b) **Set up job script:** After running this section of the code, we choose our estimated run time for the job. The default run time is 3 hours, which should cover most projects with under 75 photos. If we have more photos, we can increase the run time.

(c) **Set up parameters:** Here, we choose the quality settings for the dense cloud build.

The default settings which I've chosen are:

Preselection: 'GenericPreselection'
Quality: 'MediumQuality'
Filter: 'ModerateFiltering'
Accuracy: 'MediumAccuracy'
Adaptive Fitting: ‘True”

(d) **Create job script:** In this code cell, we create and display the job script for building the dense cloud that will be run on the Savio cluster, and we obtain the following output:

```
output:  #!/usr/bin/env python3
import PhotoScan
import time
print( "start time: ", time.time())
doc = PhotoScan.app.document
doc.open("/scratch//projectname.psx")
chunk = doc.chunk
chunk.matchPhotos(accuracy=PhotoScan.MediumAccuracy, preselection=PhotoScan.GenericPreselection)
chunk.alignCameras(adaptive_fitting=True)
## In Photoscan version 1.4.* the dense cloud generation task has been split into two parts -
## depth maps generation and dense cloud generation:
chunk.buildDepthMaps(quality=PhotoScan.MediumQuality, filter=PhotoScan.ModerateFiltering )
chunk.buildDenseCloud(point_colors = True)
doc.save("/scratch//projectname.psx")
print( "stop time: ", time.time())
estimateTime:  0:05:00
```

(e) **Run dense cloud build:** Running the code in this section submits the job to the Savio cluster with the job ID printed as output.

Note that in Photoscan version 1.4.* the dense cloud generation task has been split into two parts - depth maps generation and dense cloud generation.

Accordingly, the line

```
chunk.buildDenseCloud(quality=PhotoScan.{},  filter=PhotoScan.{} ) \n\
```

needed to be replaced by

chunk.buildDepthMaps(quality=PhotoScan.{}, filter=PhotoScan.{}) \n

chunk.buildDenseCloud(point_colors = True) \n

We obtain the following output in this section:

Execute the dense cloud build output: ['Submitted batch job 4810023']

4810023

(f) **Check job status:** Building the dense cloud is the longest-running phase of the photogrammetry process. Depending on how many photos you have (we have about 20 in this example), and the quality settings you've chosen, the job could run for anywhere from minutes to hours. We run the code box in this section to check on the status of the dense cloud build. We can run it multiple times for new updates. Here is some example output:

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         4810023     savio projectn  myashar PD       0:00      1 (Resources)
          JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         4810023     savio projectn  myashar PD       0:00      1 (Resources)
---------------------------------
JobId=4810023 JobName=projectname
   UserId=myashar(43974) GroupId=ucb(501) MCS_label=N/A
   Priority=379460 Nice=0 Account=ac_scsguest QOS=savio_normal
   JobState=PENDING Reason=Resources Dependency=(null)
   Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=00:05:00 TimeMin=N/A
   SubmitTime=2019-07-19T17:31:47 EligibleTime=2019-07-19T17:31:47
   StartTime=Unknown EndTime=Unknown Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   LastSchedEval=2019-07-19T17:32:02
   Partition=savio AllocNode:Sid=jupyter:425871
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=(null)
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,mem=62.50G,node=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=62.50G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
```

Gres=(null) Reservation=(null)

OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)

Command=/global/scratch/myashar/projectname/slurmscript.sh

WorkDir=/global/scratch/myashar/projectname

StdErr=/global/scratch/myashar/projectname/slurm-4810023.out

StdIn=/dev/null

StdOut=/global/scratch/myashar/projectname/slurm-4810023.out

Power=

During this step, the data folder in the scratch directory (/global/scratch/myashar/projectname/projectname.files) is updated.

(g) **Display log message:** As an optional step, we can run the code in this section to see the log file once the job is done. It includes technical information about the points that were found in each photo, rectifying disparities between photos, etc. Some example contents of the log file are as follows:

Enabling debugging

Ending argument loop

Singularity version: 2.6-dist

Exec'ing: /usr/libexec/singularity/cli/exec.exec

Evaluating args: '-B /global/scratch/myashar/projectname/:/scratch/ /global/scratch/myashar/container/photoscan_1_4_3.simg /scratch/execscript.

sh -platform offscreen'

VERBOSE [U=43974,P=79874] message_init() Set messagelevel to: 5

VERBOSE [U=43974,P=79874] singularity_config_parse() Initialize configuration file: /etc/singularity/singularity.conf

DEBUG [U=43974,P=79874] singularity_config_parse() Starting parse of configuration file /etc/singularity/singularity.conf

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key allow setuid = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key max loop devices = '256'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key allow pid ns = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key config passwd = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key config group = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key config resolv_conf = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key mount proc = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key mount sys = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key mount dev = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key mount devpts = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key mount home = 'no'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key mount tmp = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key mount hostfs = 'no'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key bind path = '/etc/localtime'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key bind path = '/etc/hosts'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key bind path = '/global/scratch'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key bind path = '/global/home/users'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key user bind control = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key enable overlay = 'no'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key mount slave = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key sessiondir max size = '16'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key allow container squashfs = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key allow container extfs = 'yes'

VERBOSE [U=43974,P=79874] singularity_config_parse() Got config key allow container dir = 'yes'

DEBUG   [U=43974,P=79874] singularity_config_parse() Finished parsing configuration file '/etc/singularity/singularity.conf'

DEBUG   [U=43974,P=79874] singularity_config_get_value_impl() No configuration entry found for 'always use nv'; returning default val

ue 'NULL'

…

[CPU] estimating 427x756x160 disparity using 427x756x8u tiles

timings: rectify: 0.012577 disparity: 0.376197 borders: 0.010553 filter: 0.014393 fill: 0

[CPU] estimating 326x681x192 disparity using 326x681x8u tiles

timings: rectify: 0.009837 disparity: 0.273089 borders: 0.008901 filter: 0.011632 fill: 0

Depth reconstruction devices performance:

 - 100% done by CPU

Total time: 62.825 seconds

BuildDenseCloud: point colors = 1

selected 20 cameras in 0.100879 sec

working volume: 1102x914x880

tiles: 1x1x1

selected 20 cameras in 7.8e-05 sec

preloading data... done in 0.105584 sec

filtering depth maps... done in 3.57691 sec

preloading data... done in 7.59218 sec

working volume: 1102x914x880

tiles: 1x1x1

accumulating data... done in 0.194662 sec

building point cloud... done in 0.113219 sec

2130604 points extracted

SaveProject: path = /scratch//projectname.psx

saved project in 0.734247 sec

LoadProject: path = /scratch//projectname.psx

loaded project in 0.023809 sec

stop time: 1563583287.968086

(h) **Export dense cloud:** If you want to export the dense cloud for use in other software, you can run the code in this section. If you want to just continue with the next step in the photogrammetry process, you can skip it.

The large file “photoscandemo.oc3” is created in /global/scratch/myashar/projectname/

Some of the text output from this section includes:

Points For .. 'PointsFormatOC3'
Export the points output: ['Submitted batch job 4810061']

4810061

(i) **Update project on box:** After generating the dense cloud, the project file (projectname.psx) and the data zip file (projectname.files.zip) will be much larger and uploading it to Box may take a moment. Unfortunately, I have sometimes received the following output and error message when running this part of the code, resulting in the files not being uploaded to box:

find_folder_id folder_name: myBoxFolderName

find_folder_id folderlist: [<Box Folder - 53223729428 (myBoxFolderName)>, <Box Folder - 52394687398 (myBoxFolderName_Test1)>]

Box folder id: 53223729428

project file list: [<Box File - 317022687393 (projectname.psx)>]

datazip file list: [<Box File - 319186364581 (README.docx)>, <Box File - 317017752277 (projectname.files.zip)>, <Box File - 319043117371 (slurm-generate_model_output.txt)>, <Box File - 319044387168 (slurm-build_texture.txt)>, <Box File - 319044323866

(slurm-export_photomosaic_format_output.txt)>, <Box File - 319035518579 (slurm-export_dense_cloud.txt)>, <Box File - 319043284628 (slurm-pull_images_from_Box.txt)>, <Box File - 319043305927 (slurm-build_dense_cloud.txt)>, <Box File - 319041408570 (slurm-build_mesh.txt)>]

project file id: 317022687393

data zip file list: 319186364581

begin project file update.

BoxAPIException Traceback (most recent call last)

...

BoxAPIException:

Message: Access denied - insufficient permission

Status: 403

Code: access_denied_insufficient_permissions

Request id: 1018dmg562cnfb0n

Headers: {'Content-Type': 'application/json', 'Content-Length': '217', 'Connection': 'keep-alive', 'Date': 'Sat, 20 Jul 2019 01:49:01 GMT', 'Cahis particular che-Control': 'no-cache, no-store', 'Vary': 'Accept-Encoding', 'Strict-Transport-Security': 'max-age=31536000'}

URL: https://upload.box.com/api/2.0/files/319186364581/content

Method: POST

Context info: None

Sometimes this problem can be remedied by running this code box multiple times until the upload is successful and one gets output such as the following:

find_folder_id folder_name: myBoxFolderName

find_folder_id folderlist: [<Box Folder - 53223729428 (myBoxFolderName)>, <Box Folder - 52394687398 (myBoxFolderName_Test1)>]

Box folder id: 53223729428

project file list: [<Box File - 317022687393 (projectname.psx)>]

datazip file list: [<Box File - 317017752277 (projectname.files.zip)>, <Box File - 319186364581 (README.docx)>, <Box File - 319043117371 (slurm-generate_model_output.txt)>, <Box File - 319044387168 (slurm-build_texture.txt)>, <Box File - 319044323866 (slurm-export_photomosaic_format_output.txt)>, <Box File - 319035518579 (slurm-export_dense_cloud.txt)>, <Box File - 319043284628 (slurm-pull_images_from_Box.txt)>, <Box File - 319043305927 (slurm-build_dense_cloud.txt)>, <Box File - 319041408570 (slurm-build_mesh.txt)>]

project file id: 317022687393

data zip file list: 317017752277

begin project file update.

update psx result: <Box File - 317022687393> update zip result: <Box File - 317017752277>

(j) **Dense cloud cleanup (offline):** At this point, you may want to clean up the dense point cloud by removing stray points (e.g. for objects that have not been photographed in a lightbox, where there might be artifacts from the background, etc.) This should be done using the desktop version of Photoscan on a Windows or Mac OS system (or, if possible, on the Savio visualization node).

Be sure to save the updated project file ('projectname.psx') back to the same place in Box ('myBoxFolderName') if you want to run the next processing steps through this notebook. (I have not done this in these test runs, but might be worthwhile to pursue in future work).

(11) Build Mesh

(a) First, we retrieve the project file (projectname.psx) and the data zip file (projectname.files.zip) from Box, in case it has been updated (e.g., offline dense cloud cleanup steps) and download it back to **/global/scratch/myashar/projectname/** in our example:

find_folder_id folder_name: myBoxFolderName

find_folder_id folderlist: [<Box Folder - 53223729428 (myBoxFolderName)>, <Box Folder - 52394687398 (myBoxFolderName_Test1)>]

downloading: projectname.files.zip

data zip file download complete.

downloading: projectname.psx

new file: <_io.BufferedWriter name='/global/scratch/myashar/projectname/projectname.psx'>

project file download complete.

(a) **Choose parameters:** Running the code in this section creates three sets of drop-down menus where you can choose the surface type, face count, and data source. For our test runs, we went with the default settings as follows:

Surface type: 'Arbitrary'

Face count: 'HighFaceCount'

Data source: 'DenseCloudData'

(b) **Run mesh build:** The code in this section submits the job (i.e., executes the mesh build output) on Savio (via the SLURM sbatch job submission system) and prints out the job ID, e.g.,

Execute the mesh build output: ['Submitted batch job 4814475']

4814475

(c) **Check on mesh build status:** Running the code box in this section gets the current status of the mesh build job. It can run it multiple times for the most recent status, e.g.,

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)

-------------------------------

JobId=4814475 JobName=projectname

   UserId=myashar(43974) GroupId=ucb(501) MCS_label=N/A

   Priority=387228 Nice=0 Account=ac_scsguest QOS=savio_normal

   JobState=COMPLETED Reason=None Dependency=(null)

   Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0

   RunTime=00:01:14 TimeLimit=00:30:00 TimeMin=N/A

    SubmitTime=2019-07-20T16:31:51 EligibleTime=2019-07-20T16:31:51

   StartTime=2019-07-20T16:32:38 EndTime=2019-07-20T16:33:52 Deadline=N/A

   PreemptTime=None SuspendTime=None SecsPreSuspend=0

   LastSchedEval=2019-07-20T16:32:38

   Partition=savio AllocNode:Sid=jupyter:425871

   ReqNodeList=(null) ExcNodeList=(null)

   NodeList=n0070.savio1

   BatchHost=n0070.savio1

   NumNodes=1 NumCPUs=20 NumTasks=0 CPUs/Task=1 ReqB:S:C:T=0:0:*:*

   TRES=cpu=20,mem=62.50G,node=1,billing=20

   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*

   MinCPUsNode=1 MinMemoryNode=62.50G MinTmpDiskNode=0

   Features=(null) DelayBoot=00:00:00

   Gres=(null) Reservation=(null)

   OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)

   Command=/global/scratch/myashar/projectname/slurmscript.sh

   WorkDir=/global/scratch/myashar/projectname

   StdErr=/global/scratch/myashar/projectname/slurm-4814475.out

   StdIn=/dev/null

   StdOut=/global/scratch/myashar/projectname/slurm-4814475.out

   Power=
```

(d) **Display log message:** As an optional step, you can run the code in this section to see the log file once the job is done. If the run is successful, the log file output should include something like the following:

Enabling debugging

Ending argument loop

Singularity version: 2.6-dist

Exec'ing: /usr/libexec/singularity/cli/exec.exec

Evaluating args: '-B /global/scratch/myashar/projectname/:/scratch/
/global/scratch/myashar/container/photoscan_1_4_3.simg /scratch/execscript.

sh -platform offscreen'

VERBOSE [U=43974,P=76523] message_init() Set messagelevel to: 5

VERBOSE [U=43974,P=76523] singularity_config_parse() Initialize configuration file:
/etc/singularity/singularity.conf

DEBUG [U=43974,P=76523] singularity_config_parse() Starting parse of configuration file
/etc/singularity/singularity.conf

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key allow setuid = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key max loop devices = '256'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key allow pid ns = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key config passwd = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key config group = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key config resolv_conf = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key mount proc = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key mount sys = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key mount dev = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key mount devpts = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key mount home = 'no'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key mount tmp = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key mount hostfs = 'no'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key bind path = '/etc/localtime'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key bind path = '/etc/hosts'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key bind path = '/global/scratch'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key bind path = '/global/home/users'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key user bind control = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key enable overlay = 'no'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key mount slave = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key sessiondir max size = '16'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key allow container squashfs = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key allow container extfs = 'yes'

VERBOSE [U=43974,P=76523] singularity_config_parse() Got config key allow container dir = 'yes'

DEBUG [U=43974,P=76523] singularity_config_parse() Finished parsing configuration file '/etc/singularity/singularity.conf'

…

DEBUG [U=43974,P=76324] singularity_registry_get() Returning value from registry: 'COMMAND' = 'exec'

LOG [U=43974,P=76324] main() USER=myashar, IMAGE='photoscan_1_4_3.simg', COMMAND='exec'

DEBUG [U=43974,P=76324] action_exec() Checking for: /.singularity.d/actions/exec

VERBOSE [U=43974,P=76324] action_exec() Exec'ing /.singularity.d/actions/exec

start time: 1563665561.322425

LoadProject: path = /scratch//projectname.psx

loaded project in 0.229496 sec

BuildModel: surface type = Arbitrary, source data = Dense cloud, face count = High, interpolation = Enabled, vertex colors = 1

Grid size: 914 x 880 x 1102

Tree depth: 11

Tree set in 10.855s (1533218 points)

Leaves/Nodes: 10863350/12415257

Laplacian constraints set in 2.72945s

Depth[0/11]: 1

Evaluated / Got / Solved in: 0 / 4.91142e-05 / 8.39233e-05

Depth[1/11]: 8

Evaluated / Got / Solved in: 0 / 0.000180721 / 0.00268579

Depth[2/11]: 64

Evaluated / Got / Solved in: 0 / 0.000698805 / 0.00538802

Depth[3/11]: 512

Evaluated / Got / Solved in: 0 / 0.00115395 / 0.0112739

Depth[4/11]: 4096

Evaluated / Got / Solved in: 0 / 0.00407052 / 0.0332537

Depth[5/11]: 32768

```
    Evaluated / Got / Solved in: 0 / 0.0247517 / 0.0532866
Depth[6/11]: 25984
    Evaluated / Got / Solved in: 0 / 0.0123518 / 0.0969357
Depth[7/11]: 92688
    Evaluated / Got / Solved in: 0 / 0.0453432 / 0.184737
Depth[8/11]: 336200
    Evaluated / Got / Solved in: 0 / 0.144703 / 0.428614
Depth[9/11]: 1153320
    Evaluated / Got / Solved in: 0 / 0.418476 / 0.986207
Depth[10/11]: 3543680
    Evaluated / Got / Solved in: 0 / 1.21638 / 2.20622
Depth[11/11]: 7225936
    Evaluated / Got / Solved in: 0 / 2.30052 / 4.50684
Linear system solved in 13.3021s
Got Iso-value in 0.563034s
Iso-Value 1.00491
4009108 faces extracted in 15.3532s
decimating mesh (4000167 -> 306643)
processing nodes...  done in 0.041571 sec
calculating colors...  done in 0.916212 sec
SaveProject: path = /scratch//projectname.psx
saved project in 0.085596 sec
LoadProject: path = /scratch//projectname.psx
loaded project in 0.015602 sec
stop time:  1563665631.080997
```

At this point, one should also see that the data folder “projecname.files” (for example) in the project directory in the global scratch space on Savio has been updated.

(e) **Update project on box:** The code in this section returns the project file (projectname.psx) and the data zip file (projectname.files.zip) to Box so we can make additional tweaks and changes offline with the Photoscan desktop application on a Windows or Mac OS system.

One can sometimes get an error message here. If so, try running the code box again after waiting a few seconds. If the upload of these files to Box is successful, you should get output in the Jupyter notebook such as the following:

find_folder_id folder_name: myBoxFolderName

find_folder_id folderlist: [<Box Folder - 53223729428 (myBoxFolderName)>, <Box Folder - 52394687398 (myBoxFolderName_Test1)>]

Box folder id: 53223729428

project file list: [<Box File - 317022687393 (projectname.psx)>]

datazip file list: [<Box File - 317017752277 (projectname.files.zip)>, <Box File - 319186364581 (README.docx)>, <Box File - 319043117371 (slurm-generate_model_output.txt)>, <Box File - 319044387168 (slurm-build_texture.txt)>, <Box File - 319044323866 (slurm-export_photomosaic_format_output.txt)>, <Box File - 319035518579 (slurm-export_dense_cloud.txt)>, <Box File - 319043284628 (slurm-pull_images_from_Box.txt)>, <Box File - 319043305927 (slurm-build_dense_cloud.txt)>, <Box File - 319041408570 (slurm-build_mesh.txt)>]

project file id: 317022687393

data zip file list: 317017752277

begin project file update.

update psx result: <Box File - 317022687393>   update zip result: <Box File - 317017752277>

and see that the projectname.psx and projectname.files.zip files in the 'myBoxFolderName' folder (for example) on Box have been updated,

(f) **Mesh cleanup (offline):** If you need to add marker points in the Photoscan project file (.psx), then you can download the latest project file (projectname.psx and projectname.files.zip) from Box to your Windows or Mac OS system, make the changes with the desktop Photoscan application, and upload the edited .psx etc file back to Box.

(12) Build Texture

(a) First, we run the code box here to retrieve the project file (projectname.psx and projectname.files.zip) from Box in case it has been updated (after further processing with the Photoscan desktop application on a Windows or Mac OS system). If successful, we would should receive output such as the following:

find_folder_id folder_name: myBoxFolderName

find_folder_id folderlist: [<Box Folder - 53223729428 (myBoxFolderName)>, <Box Folder - 52394687398 (myBoxFolderName_Test1)>]

downloading: projectname.files.zip

data zip file download complete.

downloading: projectname.psx

new file: <_io.BufferedWriter name='/global/scratch/myashar/projectname/projectname.psx'>

project file download complete.

(b) **Choose parameters:** Choose the data source, and specify a coordinate system if desired, after running the code box in this section. For our test runs, we chose the default settings as follows:

UV Mapping: ‘GenericMapping’

Data source: ‘ModelData’

Color Correction: ‘True’

Data Source: ‘ModelData;

When running the next code box, I received the following error message:

Traceback (most recent call last):

File "/scratch/commandscript.sh", line 9, in <module>

chunk.buildTexture(color_correction=True)

TypeError: 'color_correction' is an invalid keyword argument for this function

The error was due to the fact that **the 'color_correction' argument has been removed from the Chunk.buildTexture() method starting with Photoscan Version 1.4.0.**

Accordingly, the following modifications (highlighted in yellow below) needed to be made to the code in this section:

```
template = '#!/usr/bin/env python3 \n\
import PhotoScan \n\
import time \n\
print( "start time: ", time.time()) \n\
doc = PhotoScan.app.document \n\
doc.open(\"{}\") \n\
chunk = doc.chunk \n\
chunk.buildUV(mapping=PhotoScan.{}) \n\
chunk.calibrateColors( source_data=PhotoScan.DataSource.{}, color_balance={} ) \n\
chunk.buildTexture() \n\
doc.save(\"{}\") \n\
doc.open(\"{}\") \n\
chunk = doc.chunk \n\
chunk.buildOrthomosaic( surface=PhotoScan.DataSource.{} ) \n\
doc.save(\"{}\") \n\
```

```
print( "stop time: ", time.time()) \n'

output = template.format(singularitymountfolder + projectFile, gmchoice.value,
ds2choice.value, ccchoice.value, singularitymountfolder + projectFile,
singularitymountfolder + projectFile, ds2choice.value, singularitymountfolder +
projectFile)

print('output:', output)

with open(commandScript, 'w') as f:
    f.write(output)
```

Once these corrections were made, the code ran successfully, and we obtained output within the Jupyter Notebook as follows:

```
output: #!/usr/bin/env python3

import PhotoScan

import time

print( "start time: ", time.time())

doc = PhotoScan.app.document

doc.open("/scratch//projectname.psx")

chunk = doc.chunk

chunk.buildUV(mapping=PhotoScan.GenericMapping)

chunk.calibrateColors( source_data=PhotoScan.DataSource.ModelData, color_balance=True )

chunk.buildTexture()

doc.save("/scratch//projectname.psx")

doc.open("/scratch//projectname.psx")

chunk = doc.chunk

chunk.buildOrthomosaic( surface=PhotoScan.DataSource.ModelData )

doc.save("/scratch//projectname.psx")

print( "stop time: ", time.time())
```

(c) **Run texture build:** The code in this section submits the job (i.e., executes the texture and ortho-mosaic build output) on Savio (via the SLURM sbatch job submission system) and prints out the job ID, e.g.,

```
texture and orthomosaic build output:  ['Submitted batch job 4814495']
```

4814495

(d) **Check on texture build status:** Running the code box in this section gets the current status of the texture build job. It can be run multiple times to get the most recent status, e.g.,

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         4814495     savio projectn  myashar PD       0:00      1 (Priority)
--------------------------------
JobId=4814495 JobName=projectname
   UserId=myashar(43974) GroupId=ucb(501) MCS_label=N/A
   Priority=387228 Nice=0 Account=ac_scsguest QOS=savio_normal
   JobState=PENDING Reason=Priority Dependency=(null)
   Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=00:30:00 TimeMin=N/A
   SubmitTime=2019-07-20T17:29:35 EligibleTime=2019-07-20T17:29:35
   StartTime=2019-07-20T17:33:40 EndTime=2019-07-20T18:03:40 Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   LastSchedEval=2019-07-20T17:31:17
   Partition=savio AllocNode:Sid=jupyter:425871
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=(null) SchedNodeList=n0067.savio1
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,mem=62.50G,node=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=62.50G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   Gres=(null) Reservation=(null)
   OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
   Command=/global/scratch/myashar/projectname/slurmscript.sh
   WorkDir=/global/scratch/myashar/projectname
   StdErr=/global/scratch/myashar/projectname/slurm-4814495.out
   StdIn=/dev/null
   StdOut=/global/scratch/myashar/projectname/slurm-4814495.out
   Power=
```

(e) **Display log message:** As an optional step, you can run the code in this section (which I added) to see the log file once the job is done. If the run is successful, the log file output should include something like the following:

```
Enabling debugging
Ending argument loop
Singularity version: 2.6-dist
Exec'ing: /usr/libexec/singularity/cli/exec.exec
Evaluating args: '-B /global/scratch/myashar/projectname/:/scratch/
/global/scratch/myashar/container/photoscan_1_4_3.simg /scratch/execscript.
sh -platform offscreen'
…..
DEBUG   [U=43974,P=103815] singularity_registry_get()          Returning value from registry: 'COMMAND' =
'exec'
LOG     [U=43974,P=103815] main()                              USER=myashar, IMAGE='photoscan_1_4_3.simg',
COMMAND='exec'
DEBUG   [U=43974,P=103815] action_exec()                       Checking for: /.singularity.d/actions/exec
VERBOSE [U=43974,P=103815] action_exec()                       Exec'ing /.singularity.d/actions/exec
start time:  1563669264.2668166
LoadProject: path = /scratch//projectname.psx
loaded project in 0.108683 sec
BuildUV: mapping mode = Generic, texture count = 1
Packing 50 charts
Parameterizing done in 39.0857 sec
CalibrateColors: source data = Model
collecting control points... collected 137567 points in 1.08052 sec
analyzing model... ********** done in 0.23541 sec
analyzing 20 images... ********** done in 0.706361 sec
estimating corrections...
disabled 1408 out of 132548 points (rms error 0.0722249)
adjusting: xxxxxxx 0.0682595 -> 0.0555103
adjusting: xxxxxxxx 0.0695461 -> 0.0553696
adjusting: xxxxxxxx 0.068948 -> 0.0542193
disabled 3740 out of 132548 points (rms error 0.0537647)
adjusting: xxxxxx 0.0491485 -> 0.0482608
adjusting: xxxxxx 0.0487021 -> 0.0477439
adjusting: xxxxxxx 0.0474039 -> 0.0463474
disabled 4915 out of 132548 points (rms error 0.0468758)
adjusting: xxxxxx 0.0459288 -> 0.0455199
adjusting: xxxxxx 0.0453782 -> 0.044954
adjusting: xxxxxx 0.0439151 -> 0.043448
finished RBA in 20.3747 seconds
BuildTexture: blending mode = Mosaic, texture size = 2048, fill holes = 1, ghosting filter = 1
calculating mesh connectivity... done in 0.102164 sec
detecting outliers... ********** done in 9.46008 sec
estimating quality... ********** done in 11.3318 sec
blending textures...  ********** done in 11.998 sec
postprocessing atlas... done in 0.385161 sec
SaveProject: path = /scratch//projectname.psx
saved project in 0.113171 sec
LoadProject: path = /scratch//projectname.psx
loaded project in 0.012539 sec
LoadProject: path = /scratch//projectname.psx
```

```
loaded project in 0.010909 sec
BuildOrthomosaic: projection = Planar, surface = Mesh, blending mode = Mosaic, resolution = 0
tessellating mesh... done (306642 -> 306642 faces)
generating 2899x3459 orthomosaic (6 levels, 0.000348132 resolution)
selected 20 cameras
Orthorectifying 20 images
IMG_0327: 2899x3459 -> 2676x3163
IMG_0326: 2899x3459 -> 2090x3184
IMG_0330: 2899x3408 -> 2864x3053
IMG_0328: 2899x3459 -> 2888x3112
IMG_0329: 2899x3459 -> 2890x3021
IMG_0331: 2899x3408 -> 2836x3112
IMG_0332: 2899x3408 -> 2844x3176
IMG_0333: 2899x3408 -> 2844x3176
IMG_0335: 2889x3408 -> 2011x3176
IMG_0346: 2899x3459 -> 2458x3266
IMG_0338: 2899x3459 -> 2794x3151
IMG_0339: 2899x3459 -> 2858x3150
IMG_0340: 2899x3459 -> 2888x3202
IMG_0342: 2899x3459 -> 2899x3266
IMG_0341: 2899x3459 -> 2899x3266
IMG_0325: 2899x3459 -> 2029x3202
IMG_0334: 2899x3408 -> 2844x3176
IMG_0324: 2899x3459 -> 1897x3266
IMG_0336: 2889x3408 -> 2050x3112
IMG_0337: 2899x3408 -> 2229x3176
Finished orthorectification in 9.95988 sec
selected 1 tiles
selected 1 tiles
20 of 20 processed in 1.28607 sec
partition updated in 1.49285 sec
selected 2 tiles
loaded partition in 0.078006 sec
boundaries extracted in 0.025828 sec
18 images blended in 2.27716 sec
orthomosaic updated in 2.84187 sec
SaveProject: path = /scratch//projectname.psx
saved project in 0.015501 sec
LoadProject: path = /scratch//projectname.psx
loaded project in 0.019046 sec
stop time:  1563669376.916963
```

(f) **Update project on box:** The code in this section returns the project file (projectname.psx) and the data zip file (projectname.files.zip) to Box One can sometimes get an error message here. If so, try running the code box again after waiting a few seconds. If the upload of these files to Box is successful, you should get output in the Jupyter notebook such as the following:

```
find_folder_id folder_name:  myBoxFolderName

find_folder_id folderlist:  [<Box Folder - 53223729428 (myBoxFolderName)>, <Box Folder - 52394687398 (myBoxFolderName_Test1)>]
```

Box folder id: 53223729428

project file list: [<Box File - 317022687393 (projectname.psx)>]

datazip file list: [<Box File - 317017752277 (projectname.files.zip)>, <Box File - 319186364581 (README.docx)>, <Box File - 319043117371 (slurm-generate_model_output.txt)>, <Box File - 319044387168 (slurm-build_texture.txt)>, <Box File - 319044323866 (slurm-export_photomosaic_format_output.txt)>, <Box File - 319035518579 (slurm-export_dense_cloud.txt)>, <Box File - 319043284628 (slurm-pull_images_from_Box.txt)>, <Box File - 319043305927 (slurm-build_dense_cloud.txt)>, <Box File - 319041408570 (slurm-build_mesh.txt)>]

project file id: 317022687393

data zip file list: 317017752277

begin project file update.

update psx result: <Box File - 317022687393>   update zip result: <Box File - 317017752277>

(13) Export Othomosaic Results

(a) **Export the orthomosaic -- choose parameters:** We use the default image format option when running this part of the code:

Image Format: '**ImageFormatTIFF**'
(b) **Submit job on Savio to export the orthomosaic format output:**
In the next section of code we submit the SLURM job on Savio to export the orthomosaic format output, e.g.,

Export the orthomosaic format output: ['Submitted batch job 4814740']
4814740

and we check the job status, e.g.,

JOBID PARTITION   NAME   USER ST   TIME NODES NODELIST(REASON)

JobId=4814740 JobName=projectname

  UserId=myashar(43974) GroupId=ucb(501) MCS_label=N/A

  Priority=1001000 Nice=0 Account=ac_scsguest QOS=savio_normal

  JobState=COMPLETED Reason=None Dependency=(null)

  Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0

  RunTime=00:00:06 TimeLimit=00:30:00 TimeMin=N/A

  SubmitTime=2019-07-21T00:23:20 EligibleTime=2019-07-21T00:23:20

  StartTime=2019-07-21T00:23:20 EndTime=2019-07-21T00:23:26 Deadline=N/A

  PreemptTime=None SuspendTime=None SecsPreSuspend=0

  LastSchedEval=2019-07-21T00:23:20

  Partition=savio AllocNode:Sid=jupyter:425871

ReqNodeList=(null) ExcNodeList=(null)

NodeList=n0015.savio1

BatchHost=n0015.savio1

NumNodes=1 NumCPUs=20 NumTasks=0 CPUs/Task=1 ReqB:S:C:T=0:0:*:*

TRES=cpu=20,mem=62.50G,node=1,billing=20

Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*

MinCPUsNode=1 MinMemoryNode=62.50G MinTmpDiskNode=0

Features=(null) DelayBoot=00:00:00

Gres=(null) Reservation=(null)

OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)

Command=/global/scratch/myashar/projectname/slurmscript.sh

WorkDir=/global/scratch/myashar/projectname

StdErr=/global/scratch/myashar/projectname/slurm-4814740.out

StdIn=/dev/null

StdOut=/global/scratch/myashar/projectname/slurm-4814740.out

Power=

(c) **Display log message:** As an optional step, you can run the code in this section to see the log file of the SLURM sbatch job once the job is done. If the run is successful, the log file output should include something like the following:

Enabling debugging

Ending argument loop

Singularity version: 2.6-dist

Exec'ing: /usr/libexec/singularity/cli/exec.exec

Evaluating args: '-B /global/scratch/myashar/projectname/:/scratch/
/global/scratch/myashar/container/photoscan_1_4_3.simg /scratch/execscript.

sh -platform offscreen'

VERBOSE [U=43974,P=72899] message_init() Set messagelevel to: 5

VERBOSE [U=43974,P=72899] singularity_config_parse() Initialize configuration file:
/etc/singularity/singularity.conf

DEBUG [U=43974,P=72899] singularity_config_parse() Starting parse of configuration file
/etc/singularity/singularity.conf

…..

LOG [U=43974,P=72706] main() USER=myashar, IMAGE='photoscan_1_4_3.simg',
COMMAND='exec'

DEBUG [U=43974,P=72706] action_exec() Checking for: /.singularity.d/actions/exec

VERBOSE [U=43974,P=72706] action_exec() Exec'ing /.singularity.d/actions/exec

LoadProject: path = /scratch//projectname.psx

loaded project in 0.250962 sec

ExportRaster: image_format = TIFF, path = /scratch/projectname.tiff, tile_height = 0, tile_width = 0

generating 2898 x 3459 raster in 1 x 1 tiles

(d) **Move the orthomosaic file to Box:** The code in this section returns the project file (projectname.psx) and the data zip file (projectname.files.zip) to Box and moves the orthomosaic file projectname.tiff to Box as well. One can sometimes get an error message here, such as the following:

```
  3    """Callback for storing refresh tokens. (For now we ignore access tokens)."""
----> 4    with open(homeFolder,'apptoken.cfg', 'w') as f:
  5    f.write(refresh_token.strip())
TypeError: an integer is required (got type str)
```

--or--

BoxAPIException:

Message: Item with the same name already exists

Status: 409

Code: item_name_in_use

Request id: a63uqag588dpyoc7

Headers: {'Content-Type': 'application/json', 'Content-Length': '473', 'Connection': 'keep-alive', 'Date': 'Sun, 21 Jul 2019 22:48:46 GMT', 'Cache-Control': 'no-cache, no-store', 'Vary': 'Accept-Encoding', 'Strict-Transport-Security': 'max-age=31536000'}

URL: https://upload.box.com/api/2.0/files/content

Method: POST

Context info: {'conflicts': {'etag': '0', 'id': '317839145640', 'sha1': '7463dec43bcd257e9d986a1eb3684a923e126bcd', 'file_version': {'type': 'file_version', 'id': '335102768952', 'sha1': '7463dec43bcd257e9d986a1eb3684a923e126bcd'}, 'name': '**projectname.tiff**', 'type': 'file', 'sequence_id': '0'}}

If so, try running the code box again after waiting a few seconds, and/or make sure that there isn't already an orthomosaic file in Box with the same name. If that is the case, then change the name of the already-existing orthomosaic file in Box to something else first. If the upload of these files to Box is successful, you should get output in the Jupyter notebook such as the following:

find_folder_id folder_name: myBoxFolderName

find_folder_id folderlist: [<Box Folder - 53223729428 (myBoxFolderName)>, <Box Folder - 52394687398 (myBoxFolderName_Test1)>]

orthomosaic file: projectname.tiff

orthomosaic file id: 494460543876

(14) Generate the model export

(a) **Choose parameters:** We choose the default output file extensions and model formats after running the code boxes in this section, e.g.,

Model Format: 'ModelFormatOBJ'

```
output: #!/usr/bin/env python3
import PhotoScan
doc = PhotoScan.app.document
doc.open("/scratch//projectname.psx")
chunk = doc.chunk
chunk.exportModel(path="/scratch/projectname.obj", format=PhotoScan.ModelFormat.ModelFormatOBJ)
```

(b) **Submit job to generate model output, and check job status:** In the next section of code we submit the SLURM job on Savio to generate the model output, e.g.,

Generate model output: ['Submitted batch job 4816678']
4816678

and we check the job status, e.g.,

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
--------------------------------
JobId=4816678 JobName=projectname
   UserId=myashar(43974) GroupId=ucb(501) MCS_label=N/A
   Priority=304763 Nice=0 Account=ac_scsguest QOS=savio_normal
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:06 TimeLimit=00:30:00 TimeMin=N/A
   SubmitTime=2019-07-21T16:49:37 EligibleTime=2019-07-21T16:49:37
   StartTime=2019-07-21T16:49:38 EndTime=2019-07-21T16:49:44 Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   LastSchedEval=2019-07-21T16:49:38
   Partition=savio AllocNode:Sid=jupyter:425871
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=n0127.savio1
   BatchHost=n0127.savio1
   NumNodes=1 NumCPUs=20 NumTasks=0 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=20,mem=62.50G,node=1,billing=20
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=62.50G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   Gres=(null) Reservation=(null)
   OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
```

```
Command=/global/scratch/myashar/projectname/slurmscript.sh
WorkDir=/global/scratch/myashar/projectname
StdErr=/global/scratch/myashar/projectname/slurm-4816678.out
StdIn=/dev/null
StdOut=/global/scratch/myashar/projectname/slurm-4816678.out
Power=
```

When the job has been completed, the following files and directories will be created and/or updated in the scratch project directory on Savio (/global/scratch/myashar/projectname): **projectname.files, projectname.jpg, projectname.mtl, and projectname.obj.**

(c) **Display SLURM output log message:** As an optional step, you can run the code in this section to see the output log file of the SLURM sbatch job once the job is done. If the run is successful, the SLURM log file output should include something like the following:

```
Enabling debugging
Ending argument loop
Singularity version: 2.6-dist
Exec'ing: /usr/libexec/singularity/cli/exec.exec
Evaluating args: '-B /global/scratch/myashar/projectname/:/scratch/
/global/scratch/myashar/container/photoscan_1_4_3.simg /scratch/execscript.
sh -platform offscreen'
VERBOSE [U=43974,P=29120]  message_init()                    Set messagelevel to: 5
VERBOSE [U=43974,P=29120]  singularity_config_parse()        Initialize configuration file:
/etc/singularity/singularity.conf
DEBUG   [U=43974,P=29120]  singularity_config_parse()        Starting parse of configuration file
/etc/singularity/singularity.conf
…
DEBUG   [U=43974,P=29082]  singularity_registry_get()        Returning value from registry: 'COMMAND' =
'exec'
LOG     [U=43974,P=29082]  main()                    USER=myashar, IMAGE='photoscan_1_4_3.simg',
COMMAND='exec'
DEBUG   [U=43974,P=29082]  action_exec()                 Checking for: /.singularity.d/actions/exec
VERBOSE [U=43974,P=29082]  action_exec()                 Exec'ing /.singularity.d/actions/exec
LoadProject: path = /scratch//projectname.psx
loaded project in 0.807471 sec
ExportModel: format = ModelFormatOBJ, path = /scratch/projectname.obj
```

(d) **Move the OBJ and associated JPG files to Box**:

The code in this section returns the project file (projectname.psx) and the data zip file (projectname.files.zip) to Box and moves the projectname.mtl, projectname.obj, and projectname.jpg files to Box as well. One can sometimes get an error message here, such as the following:

One can sometimes get an error message here, such as the following:

```
  3     """Callback for storing refresh tokens. (For now we ignore access tokens)."""

----> 4     with open(homeFolder,'apptoken.cfg', 'w') as f:
```

```
    5     f.write(refresh_token.strip())
```

TypeError: an integer is required (got type str)

-- or --

BoxAPIException:

Message: **Item with the same name already exists**

Status: 409

Code: item_name_in_use

Request id: hyrvv4g58m36cfmf

Headers: {'Content-Type': 'application/json', 'Content-Length': '472', 'Connection': 'keep-alive', 'Date': 'Mon, 22 Jul 2019 06:43:01 GMT', 'Cache-Control': 'no-cache, no-store', 'Vary': 'Accept-Encoding', 'Strict-Transport-Security': 'max-age=31536000'}

URL: https://upload.box.com/api/2.0/files/content

Method: POST

Context info: {'conflicts': {'etag': '0', 'id': '317983307652', 'sha1': 'f8a4589ee423e74883a75f74401ca3657ac534b0', 'file_version': {'type': 'file_version', 'id': '335263606596', 'sha1': 'f8a4589ee423e74883a75f74401ca3657ac534b0'}, 'name': **'projectname.obj',** 'type': 'file', 'sequence_id': '0'}}

If so, try running the code box again after waiting a few seconds, and/or make sure that there aren't already an projectname.obj, projectname.jpg, and/or projectname.mtl files in Box (with the same names). If that is the case, then change the name of the already-existing files in Box to something else first. If the upload of these files to Box is successful, you should get output in the Jupyter notebook such as the following:

find_folder_id folder_name: myBoxFolderName

find_folder_id folderlist: [<Box Folder - 53223729428 (myBoxFolderName)>, <Box Folder - 52394687398 (myBoxFolderName_Test1)>]

obj file id: 494632908572

jpeg file id: 494626805638

obj file: projectname.obj

jpeg file: projectname.jpg

You can download the files from Box and preview using http://3dviewer.net/**.** You can also do this on the Savio visualization node. The *.obj files can be displayed with, e.g., meshlab.

(15) Viewing, processing and analyzing results with Photoscan Pro desktop application and Agisoft Viewer.

See, for example, **Kea Johnston's Photogrammetry Tutorial**

---

(16) Summary Findings and Recommendations

Note that the Savio shared file system is not likely going to be faster than your personal computer for file operations. The advantage of using a cluster like Savio is the ability to run many tasks in parallel on the CPUs with high core count and even on multiple nodes at once.

If your task is "I/O-bound", meaning more time is spent loading data from disk than is spent computing, then we might expect it to be slower on Savio than on your personal computer. To take advantage of Savio's resources, your task should be spending most of its time on parallel computation. If you have many tasks to do, you can even submit them all at once to Savio since we have hundreds of compute nodes available, and that may be faster than running them all sequentially.

There may be a number of possible advantages in the photogrammetry workflow described here over previous standard workstation-based photogrammetry workflows, including:

- Many of the photogrammetry steps are automated within the Python code in a user-friendly Jupyter notebook user-interface;
- The approach here takes advantage of a high performance computing (HPC) cluster in carrying out some of the necessary computing steps, such as building the dense cloud, running the texture and ortho- mosaic builds, etc. The advantage of using an HPC cluster like Savio is the ability to run many tasks in parallel on the CPUs with high core count and even on multiple nodes at once. Moreover, an HPC cluster system like Savio can typically process ~500 teraFLOPS (floating point operations per second).

On the other hand, there may be some weaknesses and inefficiencies in this photogrammetry workflow that can be improved upon, such as the need to move back and forth between multiple platforms when transferring the intermediate work products to Box (which involves a number of user authentication and token setup steps) and then downloading those files from Box to a laptop computer or other system for offline image processing steps using the Agisoft photoscan software, then uploading the processed intermediate files back to Box, and then transferring those files back unto the Savio system from Box.

It may be useful to consider how some of these offline (image processing/manipulation) steps can be implemented via Python code within the Jupyter notebook rather than as offline manual steps using a desktop version of Photoscan, and/or to consider whether the desktop version of Photoscan could be launched from the **BRC visualization node** (for the specific case of the Savio HPC cluster system)**.** If that were the case, then there would be much less of a need to transfer files so frequently back and forth between Savio, Box and a Mac OS or Windows system where one would need to carry out the intermediate offline image processing steps with the desktop version of Photoscan. Rather, with the use of the desktop version of Photoscan run via the BRC visualization node, all or most of the photogrammetry workflow steps (within and outside of the Jupyter notebooks) could be implemented with Savio only with little or no need for the Box authentication steps and the use of another OS on another computer during the intermediate steps in the workflow. The fact that Jupyter notebooks can also be run on the BRC visualization node, for example, is also advantageous for this proposed workflow.

Similarly, it may also be worthwhile to consider whether significant portions of the photogrammetry workflow described here, including the 'offline' image processing steps with the photoscan software, could be moved over to and implemented on a system like UC Berkeley's **Analytics Environments on Demand (AEoD) service** (and where, again, moving files to and from Box could be circumvented and/or skipped all together). This service is designed for researchers who need to run analytic software packages on a platform that is scaled up from a standard laptop or workstation, in a Windows-based environment.

Another approach that may be worthwhile for further evaluation in considering more efficient and optimal ways to carry out and implement this type of photogrammetry workflow on an HPC system is to explore the use of the Open OnDemand web portal (and possibly in combination with a visualization node as necessary). Open OnDemand is an NSF-funded open-source HPC portal with the goal of providing an easy way for system administrators to provide web access to their HPC resources. One might be able, for example, to more conveniently set up and/or develop and utilize a (plugin) interactive app for the PhotoScan software via the Open OnDemand framework on an HPC system than would otherwise be the case without the Open OnDemand portal / framework. Also, being able to transfer files via the Open OnDemand web interface might be more efficient and convenient than moving files to and from Box. The Open OnDemand web interface could also better facilitate the use of the Photoscan software on the HPC system.