(0) As I discussed with Chris Hoffman on Monday, 9/10/18, I have now worked completely through
the **PhotoscanWorkflow.ipynb** and **BoxAuthenticationBootstrap.ipynb** notebooks on the Savio cluster, 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 clould cleanup, mesh cleanup).** The names of the output files and directories are

helmet_model.mtl
helmet_model.obj
photoscandemo.oc3
projectname.files
projectname.files.zip
projectname.jpg
projectname.mtl
projectname.obj
projectname.psx
projectname.tiff

and you can find most or all of them in my shared Box folder at this URL: https://berkeley.box.com/s/7h9lm64uaseglyg14zn93tk6j6psc8vz

Note that the **helmet_model.obj** 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 documentl is described in much more detail in the shared google doc here:

https://docs.google.com/document/d/143zEgJhHhRgNspOB_hM23lc1x0OZK6qwb6VbmW5IytM/edit?usp=sharing

---

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 withsome additional notes/comments in then 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) 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 (and Jupyter) was installed, loaded, activated, and utilized.

(b) 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.

(c) 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)
```

(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 when running this part of **PhotoscanWorkflow.ipynb (Box Setup)** at a later time, and I then need to run *BoxAuthenticationBootstrap.ipynb* again.

(b) 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) As mentioned in an earlier email, in my case I 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.

(d) The user should also be aware of where the *.cfg files are actually being created in their Savio project space 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:

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

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.

(b) For issues that come up later, it may be useful for some users (thought 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 + '   ' + singularityContai
nerPath + '   ' + 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.

(5) **Fetching Images from Box**
In this section, it might be useful to un-comment / use the commented-out print statements for debugging purposes / sanity-checking.

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

(7) **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'.
This code box would need to be modified for this.

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

I believe it can be safely ignored.

(8) **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 steps can be done via python code within this Jupyter notebook rather than as an offline manual step using a desktop version of Photoscan.

(9) **Align photos and Build Dense Cloud -- Set up job script: Create job script**
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\
```

.....

(10) **Build Texture: choose parameters, run texture build:**

During this part of the workflow, 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:

```
interact(chc, txt=gmchoice)
interact(chc, txt=ds2choice)
interact(chc, txt=ccchoice)
interact(chc, txt=ds2choice)
```

```
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, singularity
mountfolder + projectFile, ds2choice.value, singularitymountfolder + projectFile)

print('output:', output)

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