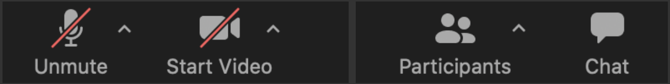
(Open-source) Photogrammetry on HPC clusters

Alex Razoumov
alex.razoumov@westdri.ca

Zoom controls

- Please mute your microphone and camera unless you have a question
- To ask questions at any time, type in Chat, or Unmute to ask via audio
  - please address chat questions to "Everyone" (not direct chat!)
- Raise your hand in Participants



- Email training@westdri.ca
- Our winter/spring training schedule `https://bit.ly/wg2024a`
  - webinars, courses, summer school at SFU on May 27–31

Oxford English Dictionary: PHOTOGRAMMETRY, n.
The technique of using photographs to ascertain measurements of what is photographed, esp. in surveying and mapping.

Nowadays, photogrammetry almost exclusively refers to the process of constructing a 3D model (= geometry of a scene) by analyzing a unordered series of photographs / videos of the same subject captured from various angles and/or with different lighting conditions

In a sense, it's the opposite of photography (which does 3D $\rightarrow$ 2D):
now recover depth information from 2D images

Proprietary photogrammetry software

- Metashape (Mac/Windows/Linux)
  - from Agisoft
  - educational discount
  - Linux version can run on HPC clusters, uses node-locked licenses
  - seems to be a popular choice in the academia
- RealityCapture: cloud or deskop (Windows only), expensive
- ReCap Photo from Autodesk
- etc.

Our approach

- Only open source
- Photogrammetry is very computationally intensive ⇒ if part of research, it is a natural application to run on HPC clusters in batch mode
  - ⇒ all command line, no GUI
  - ⇒ looking for plug and play functionality: in go images, out comes a 3D textured model
    - don't want to adjust parameters or fiddle with intermediate results
- Zero personal experience with command-line photogrammetry before this webinar
- For many of these packages, the official documentation and tutorials focus on working via a GUI ⇒ discovering actual shell commands and parameters requires digging well beyond documentation
- Do not want to compile long software dependencies and deal with complicated and obscure instructions ⇒ relying on Docker Hub containers

Creating container images on a cluster

In simpler cases you might be able to pull from Docker Hub directly on a cluster, e.g. on Cedar with online access from compute nodes:

```
cd ~/scratch
module load StdEnv/2023 apptainer/1.2.4
salloc --cpus-per-task=1 --time=0:60:0 --mem-per-cpu=3600 --account=...
mkdir $SLURM_TMPDIR/{tmp,cache}    # to avoid Lustre filesystem limitations on the host
export APPTAINER_TMPDIR=${SLURM_TMPDIR}/tmp
export APPTAINER_CACHEDIR=${SLURM_TMPDIR}/cache    # replaces `$HOME/.apptainer/cache`
apptainer pull docker://geointeractive/opensfm     # 371M file opensfm_latest.sif
apptainer pull docker://opendronemap/odm           # 540M file odm_latest.sif
apptainer pull docker://threedscan/meshroom        # create 533M file meshroom_latest.sif
cp odm_latest.sif ~/scratch/<pipeline>             # especially if in $SLURM_TMPDIR
```

... however – especially for larger images – you might run into problems, e.g. with:

- permissions for files inside the container
- Internet access from a compute node

Creating container images in a cloud VM

For larger container images you are more likely to run into problems, e.g. with permissions, Lustre filesystem limitations, Internet access from a compute node

⇒ create them as root on a Linux machine, if you have one

- you can do this inside a VM in our cloud - bash commands in the next slide
  1. (if you don't have one) apply for a cloud project – could be on Arbutus, Béluga, Graham, or Cedar
  2. inside that project, create an instance, associate a external floating IP
  3. create a volume (50GB should be sufficient) and attach it to your instance
  4. format and mount the volume in your VM
  5. install Apptainer
  6. `apptainer pull <image>.sif docker://...` into your 50GB mount

Creating container images in a cloud VM (cont.)

Example for an Ubuntu VM in Arbutus OpenStack cloud:

```
ls /dev/disk/by-id                  # get the attached disk's ID and use it in the next line
device=/dev/disk/by-id/virtio-...
sudo mkfs.ext4 $device              # format the volume
sudo mkdir -p /data                 # create a mount point
sudo mount $device /data            # mount the volume
sudo chmod og+rwX /data             # give write access to non-root

sudo apt update
sudo apt install -y software-properties-common
sudo add-apt-repository -y ppa:apptainer/ppa
sudo apt update
sudo apt install -y apptainer

export APPTAINER_TMPDIR=/data/tmp
export APPTAINER_CACHEDIR=/data/cache   # replaces default `$HOME/.apptainer/cache`
cd /data && mkdir -p tmp cache
apptainer pull docker://alicevision/meshroom:2023.3.0-av3.2.0-centos7-cuda11.3.1
scp meshroom_*.sif <username>@<cluster>.alliancecan.ca:scratch/meshroom
```

Image acquisition

- Avoid smooth surfaces and reflections: these make it much harder to reconstruct a 3D model
- Many small details make it easier to find match points, i.e.
  - OpenSfM's tutorial `https://opensfm.org/docs/using.html` demos creating an accurate 3D point cloud from only three photographs (Berlin): many intricate details
  - having a unique shape helps, as the edges will add match points
- Take 60-70 images for a detailed reconstruction
- However, more photographs $\Rightarrow$ slower processing
- Try to crop the background, if it is not related to the main object
- Important to have all pictures in-focus

84 input files

High-level pipeline view

1. Structure-from-Motion (SfM), aka sparse reconstruction
   - infer the 3D scene structure: figuring common points in 2D overlapping images, reconstructing their 3D positions along with camera poses (camera positions and orientations)

   ⇒ output is a set of calibrated cameras with a sparse point cloud

2. MultiView-Stereo (MVS), aka dense reconstruction
   - generate a dense geometric surface using the calibrated cameras + the sparse point cloud from the previous step

   ⇒ output is a textured mesh, in OBJ file format with the corresponding MTL and texture files

   2.1 dense point-cloud reconstruction (densification)
   2.2 mesh reconstruction (meshing)
   2.3 mesh refinement
   2.4 mesh texturing

High-level pipeline view

1. Structure-from-Motion (SfM), aka sparse reconstruction
   - infer the 3D scene structure: figuring common points in 2D overlapping images, reconstructing their 3D positions along with camera poses (camera positions and orientations)

   ⇒ output is a set of calibrated cameras with a sparse point cloud
2. MultiView-Stereo (MVS), aka dense reconstruction
   - generate a dense geometric surface using the calibrated cameras + the sparse point cloud from the previous step

   ⇒ output is a textured mesh, in OBJ file format with the corresponding MTL and texture files

   2.1 dense point-cloud reconstruction (densification)
   2.2 mesh reconstruction (meshing)
   2.3 mesh refinement
   2.4 mesh texturing

In practice, each of these in turn contains multiple steps that depend on the specific software chain

Structure-from-Motion (SfM)

OpenSfM package `https://opensfm.org`

- Open-source SfM library to build 3D models from images, written in Python
- Documentation `https://opensfm.org/docs`
- Source code `https://github.com/mapillary/OpenSfM`
- In addition to input images, you can also supply:
  1. `gcp_list.txt` lists Ground Control Points to help you scale+orient your 3D model
  2. `config.yaml` with additional parameters to overwrite the defaults, e.g.

```
processes: 8                     # number of threads to use
depthmap_min_consistent_views: 2 # min number of views that should reconstruct a valid point
                                 # (default 3; smaller number will give you more points)
depthmap_save_debug_files: yes   # save debug files with partial reconstruction results
feature_process_size: 1024       # photo dimensions for feature extraction
```

- A discussion of most important OpenSfM parameters in `https://github.com/OpenDroneMap/ODM/issues/769`

Running OpenSfM on a cluster

```
#!/bin/bash
#SBATCH --cpus-per-task=1
#SBATCH --time=3:0:0    # on Cedar took 2h23m for this problem on one CPU core
#SBATCH --mem-per-cpu=3600
#SBATCH --account=...
module load StdEnv/2023 apptainer/1.2.4
# run the entire OpenSfM pipeline, ~1h40m on one CPU core
find vase -mindepth 1 -maxdepth 1 -not -name 'images' | xargs /bin/rm -rf
apptainer exec -C --pwd $(pwd -P) opensfm_latest.sif /source/OpenSfM/bin/opensfm_run_all vase
```

```
cd ~/scratch/opensfm
>>> upload input images to ./vase/images
sbatch submit.sh
...
find vase -name "*.ply" 2>/dev/null
```

Underneath, this runs:

```
/source/OpenSfM/bin/opensfm extract_metadata vase
/source/OpenSfM/bin/opensfm detect_features vase
/source/OpenSfM/bin/opensfm match_features vase
/source/OpenSfM/bin/opensfm create_tracks vase
/source/OpenSfM/bin/opensfm reconstruct vase
/source/OpenSfM/bin/opensfm mesh vase
/source/OpenSfM/bin/opensfm undistort vase
/source/OpenSfM/bin/opensfm compute_depthmaps vase
```

Download to your computer:

```
scp cedar:scratch/opensfm/vase/depthmaps/merged.ply .
```

Demo OpenSfM on a training cluster for BERLIN dataset

Three input images ⇒ this takes only few mins:

```
cd ~/scratch/opensfm
module load StdEnv/2023 apptainer/1.2.4
salloc --cpus-per-task=1 --time=0:30:0 --mem-per-cpu=3600
/bin/rm -rf berlin
apptainer shell opensfm_latest.sif
cp -r /source/OpenSfM/data/berlin .    # must be on a writable filesystem, to store results
/source/OpenSfM/bin/opensfm_run_all berlin   # run the entire OpenSfM pipeline
...
find berlin -name "*.ply" 2>/dev/null | xargs ls -lh
```

Larger VASE run

Let's check out results in `~/tmp/photogrammetry/opensfm/vase`

Only a point cloud – no mesh or texture

MultiView-Stereo (MVS), aka dense reconstruction

A popular package for this is OpenMVS `https://cdcseacave.github.io`

- Open-source library to reconstruct everything from dense point clouds to textured meshes
- Somewhat sparse documentation `https://github.com/cdcseacave/openMVS/wiki`
- Source code `https://github.com/cdcseacave/openMVS`
- Instead of running it separately, let me show a pipeline built on top of it

OpenDroneMap, aka ODM

Details at https://www.opendronemap.org

- Open-source toolkit for processing aerial imagery
- Skip its georeferencing step to use it for regular photogrammetry
- Built on top of OpenSfM and OpenMVS
- Good description of the entire project
  `https://hub.docker.com/r/opendronemap/odm`

Testing OpenDroneMap on the training cluster with `salloc`

```
cd ~/scratch/odm
>>> upload input images to ./vase/images/

module load StdEnv/2023 apptainer/1.2.4
salloc --cpus-per-task=8 --time=0:120:0 --mem-per-cpu=1200
/bin/rm -rf vase/{benchmark.txt,*.json,*.txt,opensfm}
/bin/rm -rf vase/odm_{dem,filterpoints,meshing,orthophoto,report}
/bin/rm -rf vase/odm_{texturing,texturing_25d,georeferencing}
apptainer shell odm_latest.sif
python3 /code/run.py --end-with mvs_texturing $(pwd)/vase
```

Underneath, `run.py` *without flags* will attempt to run the following steps:

1. `dataset`
2. `split`
3. `merge`
4. `opensfm` - open-source Structure from Motion
5. `openmvs` - Multi-View Stereo reconstruction
6. `odm_filterpoints`
7. `odm_meshing`
8. `mvs_texturing`
9. `odm_georeferencing` - map to geographic coordinates
10. `odm_dem` - digital surface+terrain (elevation?) model
11. `odm_orthophoto` - geometrically-corrected image of the ground (high-resolution map from aerial mapping)
12. `odm_report`
13. `odm_postprocess`

Production OpenDroneMap runs on a cluster

```
#!/bin/bash
#SBATCH --cpus-per-task=16
#SBATCH --time=2:0:0
#SBATCH --mem-per-cpu=3600
#SBATCH --account=...
module load StdEnv/2023 apptainer/1.2.4
/bin/rm -rf vase/{benchmark.txt,*.json,*.txt,opensfm}
/bin/rm -rf vase/odm_{dem,filterpoints,meshing,orthophoto,report,\
                      texturing,texturing_25d,georeferencing}
apptainer exec odm_latest.sif python3 /code/run.py --end-with mvs_texturing $(pwd)/vase
```

```
cd ~/scratch/odm
upload input images to ./vase/images/
sbatch submit.sh
```

```
srun --jobid=... --pty bash
htop -u $USER -s PERCENT_CPU
```

Fine-grained control in OpenDroneMap

For more control, take a look at the output file from your job, and there you will see the exact commands with parameters, e.g.

```
$ grep running slurm-<jobID>.out
...
[INFO]    running renderdem "/scratch/razoumov/odm/vase/odm_filterpoints/point_cloud.ply"
  --outdir "/scratch/razoumov/odm/vase/odm_meshing/tmp" --output-type max
  --radiuses 0.031415926535897934,0.044428829381583667,0.06283185307179588
  --resolution 0.02 --max-tiles 0 --decimation 1 --classification -1
  --tile-size 4096 --force
...
```

Postprocessing

Download to your computer:

```
cd ~/tmp/photogrammetry/odm/vase

f1=odm_filterpoints/point_cloud.ply                    # preliminary point cloud
f2=opensfm/undistorted/openmvs/scene_dense.ply         # dense point cloud
f3=odm_meshing/odm_mesh.ply                            # polygonal mesh
f4=odm_texturing/odm_textured_model_geo.obj            # textured mesh (pointing to materials)
f5=odm_texturing/odm_textured_model_geo.mtl            # and its texture
                                     # (material definitions pointing to PNGs)
f6=odm_texturing/"odm_textured_model_geo_material*.png"    # material PNGs

scp cedar:/scratch/razoumov/odm/vase/{$f1,$f2,$f3,$f4,$f5,$f6} .
```

Let's check out results in `~/tmp/photogrammetry/odm/vase`

- open `*.ply` point clouds using ParaView's PDAL reader
- open `*.ply` meshes using ParaView's PLY reader
- open `odm_textured_model_geo.obj` in Meshlab (next slide)

Postprocessing in Meshlab

1. `cp odm_textured_model_geo.obj clip.obj`
2. Open `clip.obj` in Meshlab
3. In the toolbar: Select Vertices, Delete Selected Vertices, CMD-backspace
4. File | Export Mesh... (will be written back to clip.obj)
5. To pan: CMD-mouse

While you can load an OBJ file into ParaView, it lets you apply only one texture per polygonal mesh. Since ODM returns a single mesh with many textures, displaying it in ParaView would require splitting the original mesh into a set of meshes (and applying a separate texture to each) – possible but quite labour-intensive.

Meshroom

- Open-source 3D Reconstruction Software `https://alicevision.org/#meshroom` `https://github.com/alicevision/meshroom`
- Supported by the AliceVision Association, a non-profit organization whose goal is to democratize 3D digitization technologies from photographs
- Built on top of OpenSfM
- An NVIDIA CUDA-enabled GPU is recommended
  - without a supported NVIDIA GPU, only "Draft Meshing" from SfM step can be used for dense 3D reconstruction (very poor quality)
  - only one of 12 steps uses GPU

Meshroom on Docker Hub

- Search for Meshroom on Docker Hub `https://hub.docker.com/search?q=meshroom`
- Official recent versions are problematic:
  - checked alicevision/meshroom:2023.3.0-av3.2.0-centos7-cuda11.3.1 and alicevision/meshroom:2023.2.0-av3.1.0-centos7-cuda11.3.1
  - dense scene reconstruction (DepthMap step) crashes with

```
Assertion `row >= 0 && row < rows() && col >= 0 && col < cols()' failed
```

    - seems like the binary was compiled with debug turned on ... can't use them as is in Apptainer
- Instead, I am using an older version `docker://threedscan/meshroom`
  - compact 533M SIF file; official versions are 4.2GB and larger
  - `meshroom_photogrammetry`, has since been renamed to `meshroom_batch`
  - great experience overall!

Running the entire pipeline in a single step

- Running on a cluster's CPU partition, you will get "No CUDA-Enabled GPU" and the code will stop ⇒ need to submit a GPU job
  - there is a way to override this, but I don't recommend it, as results will be very poor

```
#!/bin/bash
#SBATCH --cpus-per-task=8 --time=1:0:0 --mem-per-cpu=3600 --gpus-per-node=1 --account=...
module load StdEnv/2023 apptainer/1.2.4
cat << EOF > run.py
nvidia-smi
export LC_ALL=C   # for some reason the executable requires localization settings
mkdir -p results && /bin/rm -rf results/*
meshroom_photogrammetry --input vase/images/ --output results
EOF
chmod u+x run.py
apptainer exec --nv meshroom_latest.sif ./run.py
tar cvfz results.tar.gz results $(find /tmp/MeshroomCache/ -name "*log" -o -name "status")
```

```
cd ~/scratch/meshroom
>>> upload input images to ./vase/images/
sbatch submit.sh
```

```
srun --jobid=... --pty bash
htop -u $USER -s PERCENT_CPU
watch -n 3 nvidia-smi   # during step 7 (DepthMap)
```

```
scp cedar:scratch/meshroom/results.tar.gz .
```

Timing

All wallclock times

1	CameraInit	0.123s
2	FeatureExtraction	56s
3	ImageMatching	0.068s
4	FeatureMatching	25s
5	StructureFromMotion	594s
6	PrepareDenseScene	6.5s
7	DepthMap (requires CUDA)	688s
8	DepthMapFilter	189s
9	Meshing	290s
10	MeshFiltering	3.5s
11	Texturing	100s
12	Publish	59s
	Total	33.5m

Look for “elapsedTime” and “Task done” in the logs.

Problem: only 1/3 of the wallclock time was spent on the GPU ...

Splitting the entire pipeline into three steps: step 1

1. Split the workflow into three steps: CPU only + GPU part + CPU only
2. Move output from `/tmp/MeshroomCache` to `./tmp/MeshroomCache`

```
#!/bin/bash
#SBATCH --cpus-per-task=16 --time=0:30:0 --mem-per-cpu=3600 --account=...
module load StdEnv/2023 apptainer/1.2.4
cat << EOF > run.py
export LC_ALL=C
export details="--input vase/images/ --output results --cache $(pwd)/tmp/MeshroomCache"
mkdir -p results && /bin/rm -rf results/*
meshroom_photogrammetry ${details} --toNode CameraInit
meshroom_photogrammetry ${details} --toNode FeatureExtraction
meshroom_photogrammetry ${details} --toNode ImageMatching
meshroom_photogrammetry ${details} --toNode FeatureMatching
meshroom_photogrammetry ${details} --toNode StructureFromMotion
meshroom_photogrammetry ${details} --toNode PrepareDenseScene
EOF
chmod u+x run.py
apptainer exec --nv meshroom_latest.sif ./run.py
```

Splitting the entire pipeline into three steps: step 2

```
#!/bin/bash
#SBATCH --cpus-per-task=8 --time=0:30:0 --mem-per-cpu=3600 --gpus-per-node=1 --account=...
module load StdEnv/2023 apptainer/1.2.4
cat << EOF > run.py
nvidia-smi
export LC_ALL=C
export details="--input vase/images/ --output results --cache $(pwd)/tmp/MeshroomCache"
meshroom_photogrammetry ${details} --toNode DepthMap_1   # requires CUDA
EOF
chmod u+x run.py
apptainer exec --nv meshroom_latest.sif ./run.py
```

Splitting the entire pipeline into three steps: step 3

```
#!/bin/bash
#SBATCH --cpus-per-task=16 --time=0:30:0 --mem-per-cpu=3600 --account=...
module load StdEnv/2023 apptainer/1.2.4
cat << EOF > run.py
export LC_ALL=C
export details="--input vase/images/ --output results --cache $(pwd)/tmp/MeshroomCache"
meshroom_photogrammetry ${details} --toNode DepthMapFilter
meshroom_photogrammetry ${details} --toNode Meshing
meshroom_photogrammetry ${details} --toNode MeshFiltering
meshroom_photogrammetry ${details} --toNode Texturing
meshroom_photogrammetry ${details} --toNode Publish
EOF
chmod u+x run.py
apptainer exec --nv meshroom_latest.sif ./run.py
tar cvfz results.tar.gz results $(find tmp/MeshroomCache/ -name "*log" -o -name "status")
```

Fine-grained control in Meshroom

- For more control, check out a contributed pipeline
  `https://github.com/davidmoncas/meshroom_CLI`
- They replace individual steps with direct calls to `aliceVision/` commands with many flags each, and breaking many individual steps into further sub-steps, e.g.

```
meshroom_photogrammetry --input vase/images/ --output results \
                        --cache $(pwd)/tmp/MeshroomCache --toNode CameraInit
```

becomes

```
mkdir -p 1_CameraInit
/opt/Meshroom-2019.2.0/aliceVision/bin/aliceVision_cameraInit \
    --imageFolder "vase/images/" \
    --sensorDatabase "/opt/Meshroom-2019.2.0/aliceVision/share/aliceVision/cameraSensors.db" \
    --output "1_CameraInit/cameraInit.sfm" --defaultFieldOfView 45 \
    --allowSingleView 1 --verboseLevel "error"
```

Let's check out results in `~/tmp/photogrammetry/meshroom/vase`

Summary

- In production runs, replace `salloc` with `sbatch` and write Slurm submission scripts
- For many of these packages, the official documentation and tutorials focus on working via a GUI, and discovering actual shell commands and parameters requires digging well beyond documentation
- Hopefully, this presentation fills in that gap if you want to run these tools via command line on a cluster
- Other open-source projects to check: COLMAP, MicMac, MVE, OpenMVG, VisualSFM, Regard3D (played with it few years ago in the GUI)
- Free and commercial photogrammetry software review (updated for 2021)

Questions?