BundleTrack (2021)

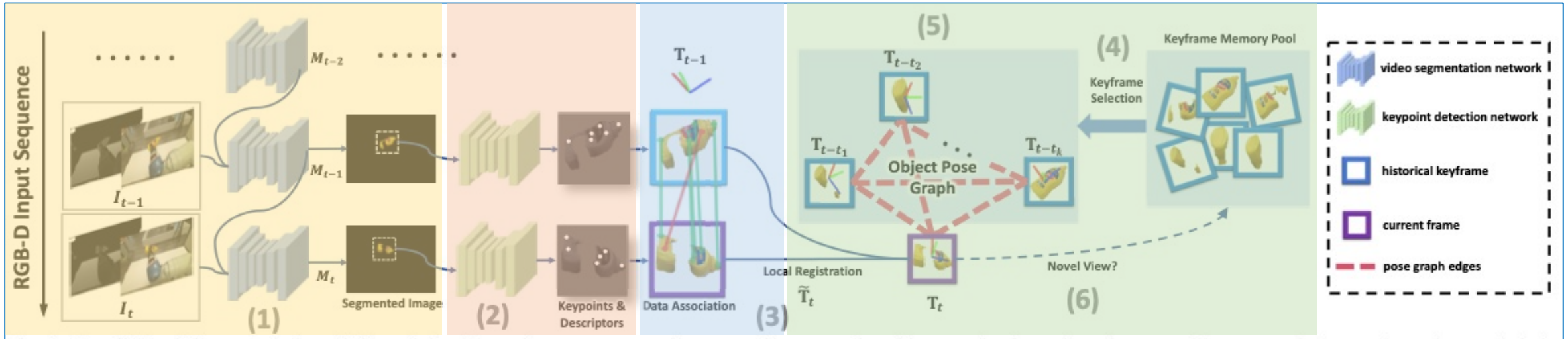


Fig. 2: *BundleTrack* framework from left to right: (1) an image segmentation network returns the object mask given the prior one; (2) a network detects keypoints and their descriptors; (3) keypoints are matched and coarse registration is performed between consecutive frames to estimate an initial relative transform $\tilde{\mathbf{T}}_t$; (4) keyframes are selected from a memory pool to participate in the pose graph optimization; (5) online pose graph optimization outputs a refined spatiotemporal consistent pose $\mathbf{T}_t$; and (6) the latest frame is included in the memory pool, if it is a novel view to enrich diversity.

1) Image segmentation
2) Keypoint detection
3) Data association to get initial coarse estimate
4) Object Pose Graph of meaningful Keyframes to compute an optimized pose for the current timestamp

BundleTrack (2021)

- Code does not run!
- Set up lightly explained

BundleSDF (2023)

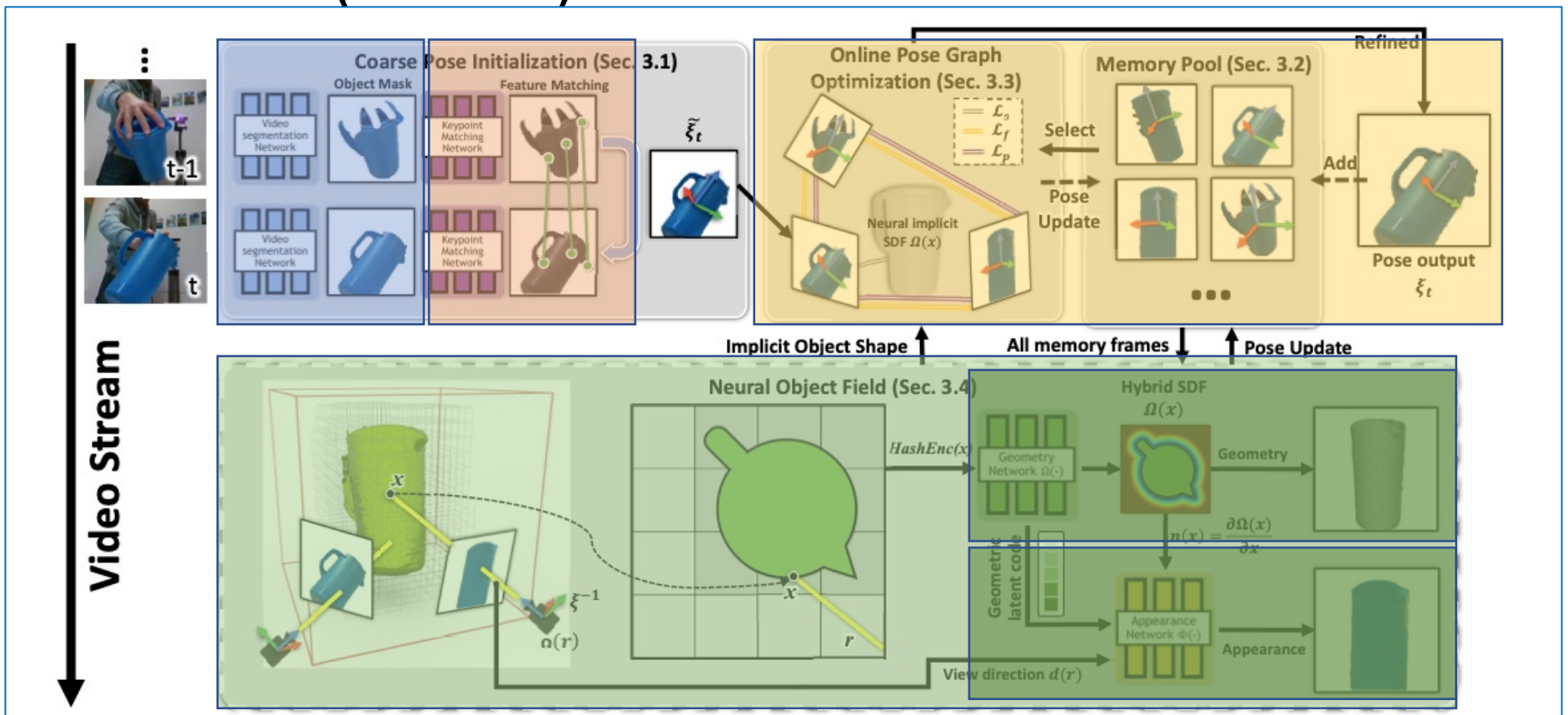


Figure 2. Framework overview. First, features are matched between consecutive segmented images, to obtain a coarse pose estimate (Sec. 3.1). Some of these posed frames are stored in a memory pool, to be used and refined later (Sec. 3.2). A pose graph is dynamically created from a subset of the memory pool (Sec. 3.3); online optimization refines all the poses in the graph jointly with the current pose. These updated poses are then stored back in the memory pool. Finally, all the posed frames in the memory pool are used to learn a Neural Object Field (in a separate thread) that models both geometry and visual texture (Sec. 3.4) of the object, while adjusting their previously estimated poses.

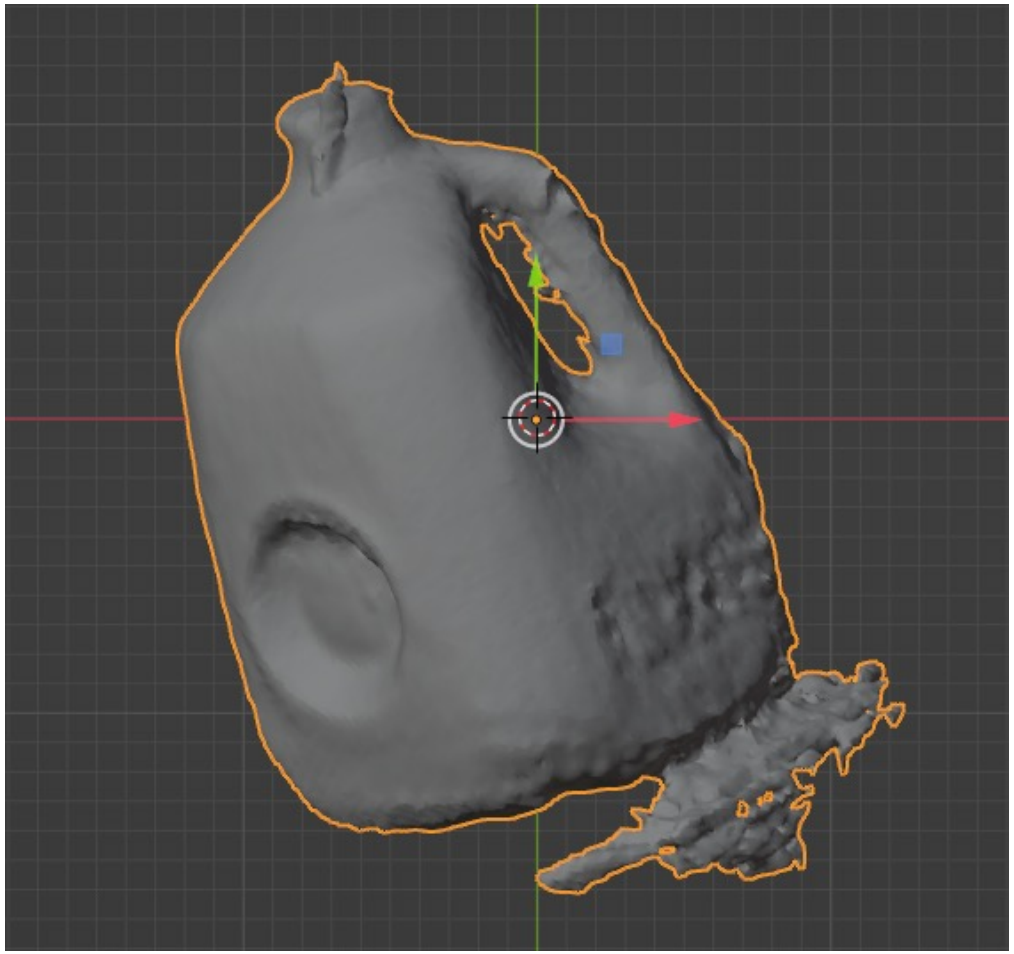
BundleTrack (2021)

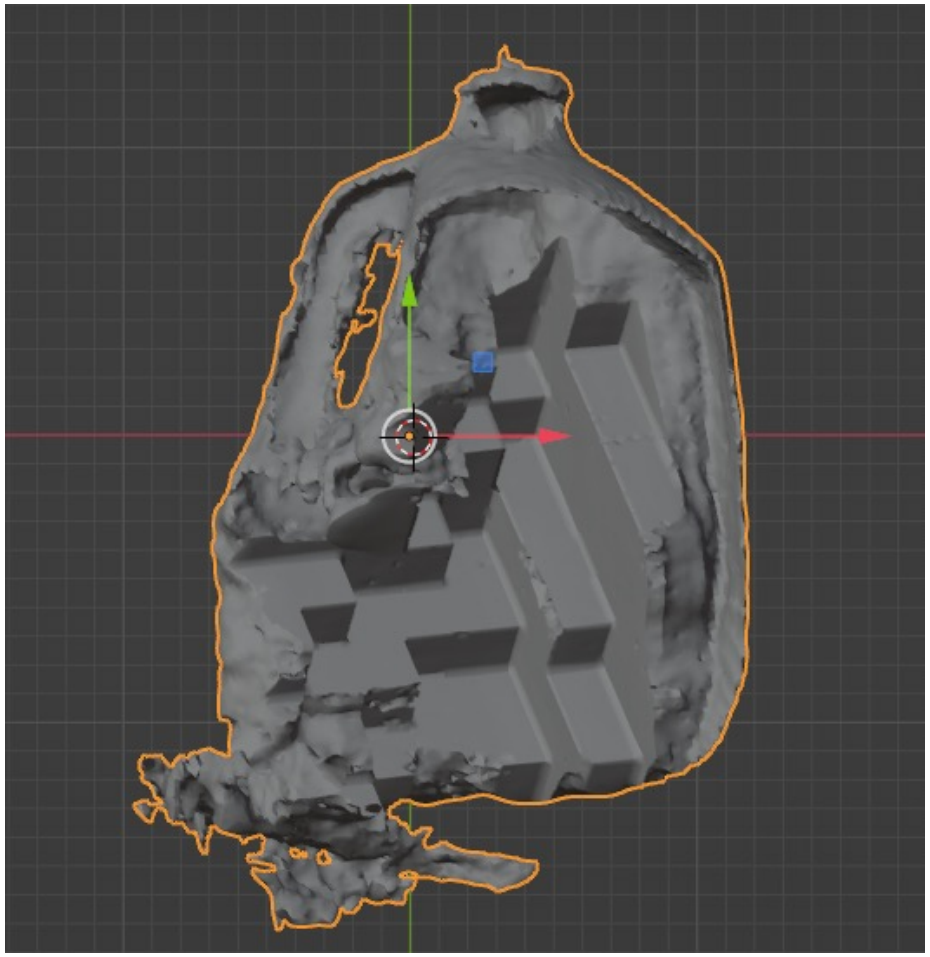
- Code runs!

BundleTrack (2021)

- Code runs!
- Milk film: CUDA out of memory
  - 7.79 GB total capacity; 2.63 GB already allocated; 219 MB free; 3.31 GB reserved in total by PyTorch

BundleTrack (2021)

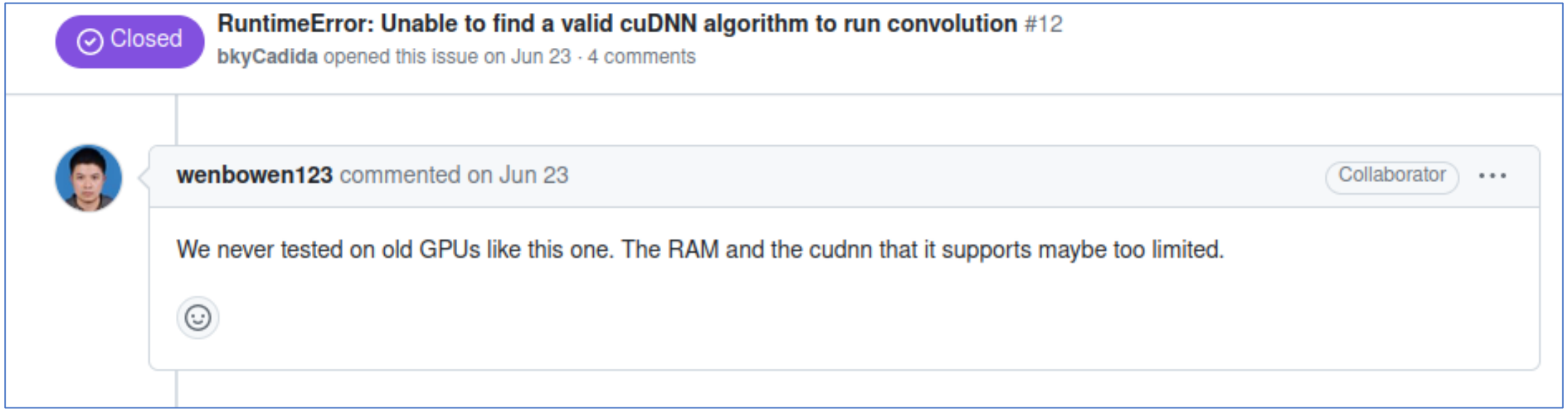




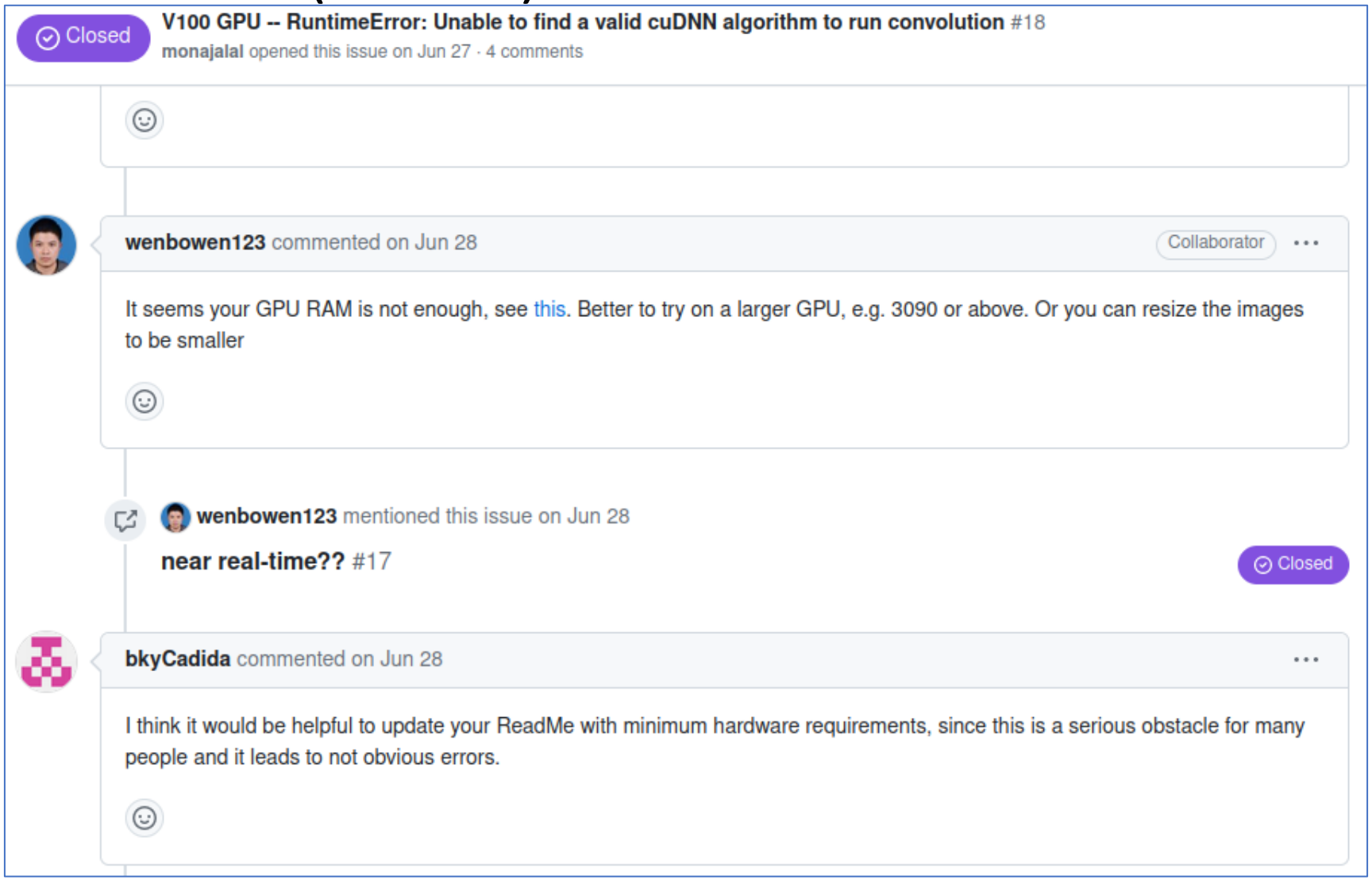
BundleTrack (2021)

- Code runs!
- Milk film: CUDA out of memory
  - 7.79 GB total capacity; 2.63 GB already allocated; 219 MB free; 3.31 GB reserved in total by PyTorch
- HO3D dataset: Unable to find a valid cuDNN algorithm to run convolution

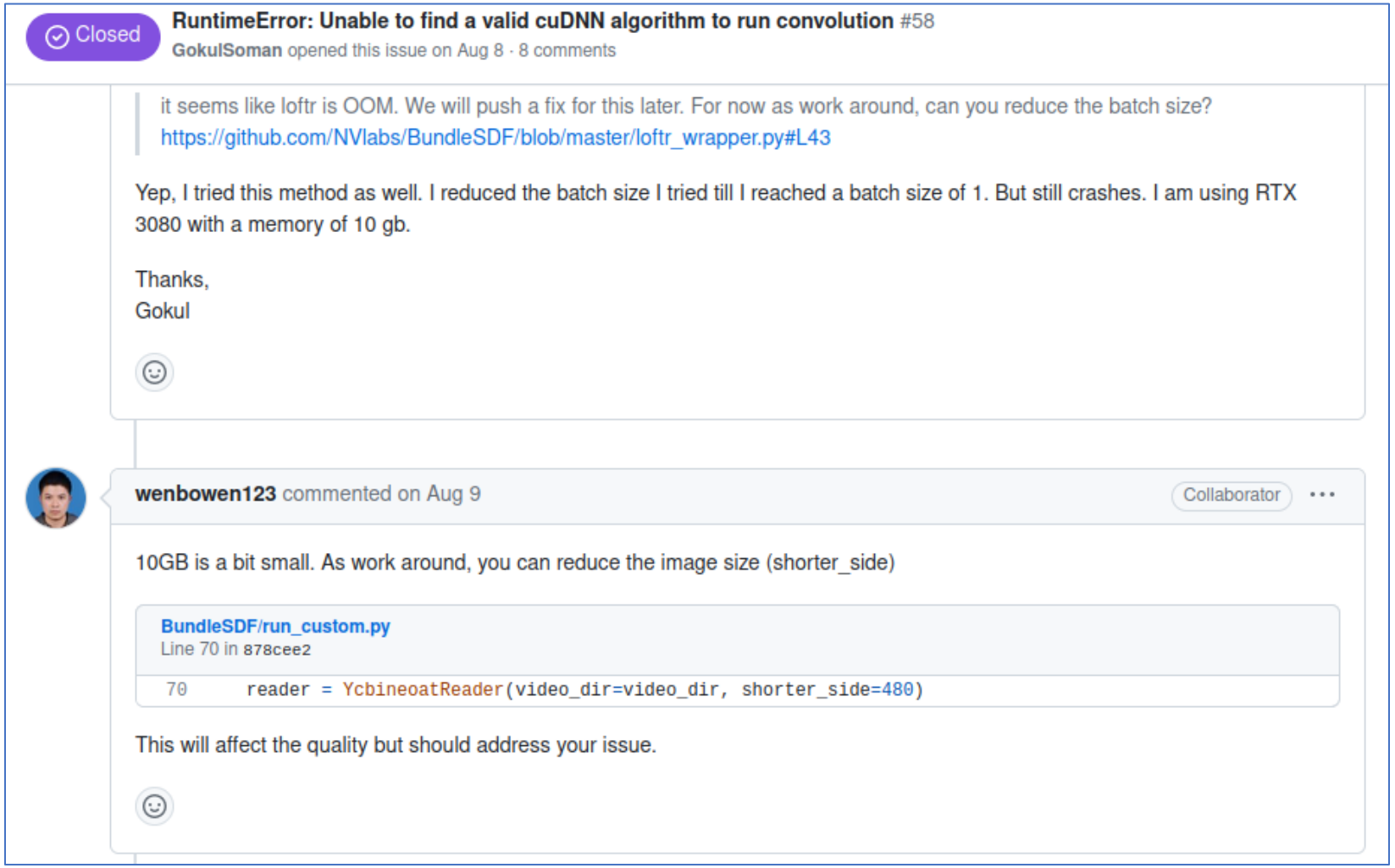
BundleTrack (2021)



BundleTrack (2021)



BundleTrack (2021)



Closed **RuntimeError: Unable to find a valid cuDNN algorithm to run convolution** #58

GokulSoman opened this issue on Aug 8 · 8 comments

> it seems like loftr is OOM. We will push a fix for this later. For now as work around, can you reduce the batch size?
> https://github.com/NVlabs/BundleSDF/blob/master/loftr_wrapper.py#L43

Yep, I tried this method as well. I reduced the batch size I tried till I reached a batch size of 1. But still crashes. I am using RTX 3080 with a memory of 10 gb.

Thanks,
Gokul

wenbowen123 commented on Aug 9 — Collaborator

10GB is a bit small. As work around, you can reduce the image size (shorter_side)

BundleSDF/run_custom.py
Line 70 in `878cee2`

```
70      reader = YcbineoatReader(video_dir=video_dir, shorter_side=480)
```

This will affect the quality but should address your issue.

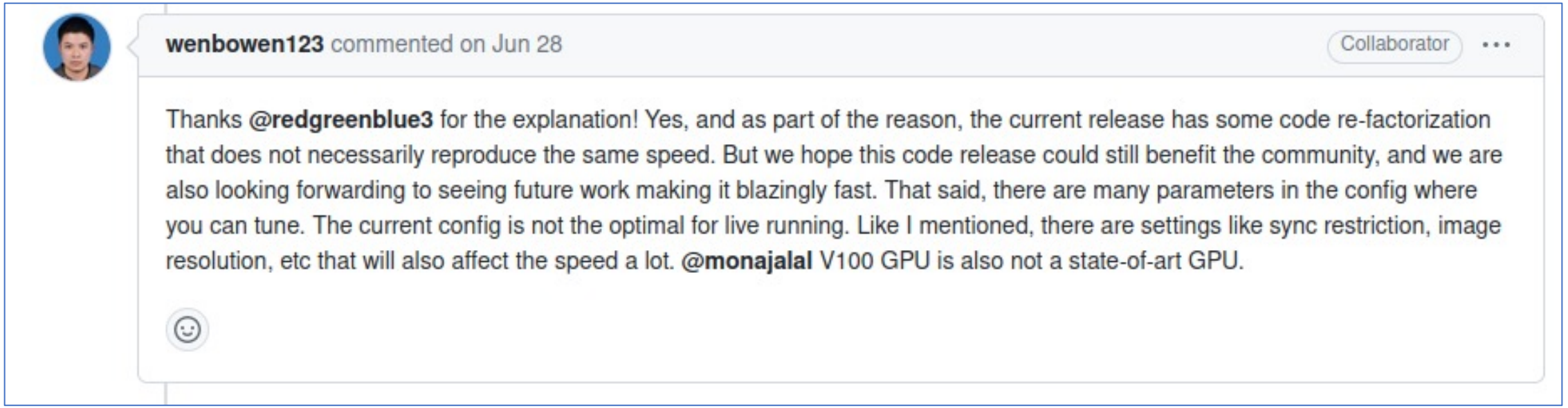
BundleTrack (2021)

- Code runs!
- Milk film: CUDA out of memory
  - 7.79 GB total capacity; 2.63 GB already allocated; 219 MB free; 3.31 GB reserved in total by PyTorch
- HO3D dataset: Unable to find a valid cuDNN algorithm to run convolution
  - Changing batch_size = 16 in loftr_wrapper takes me to CUDA out of memory

BundleTrack (2021)

- Code runs!
- Milk film: CUDA out of memory
  - 7.79 GB total capacity; 2.63 GB already allocated; 219 MB free; 3.31 GB reserved in total by PyTorch
- HO3D dataset: Unable to find a valid cuDNN algorithm to run convolution
  - Changing batch_size = 16 in loftr_wrapper takes me to CUDA out of memory
- Near real-time?
  - Every ca. 4 frames trains the pose network?
  - Extremely slow, 5 minutes for 100 frames and then crashes

BundleTrack (2021)



wenbowen123 commented on Jun 28 Collaborator

Thanks **@redgreenblue3** for the explanation! Yes, and as part of the reason, the current release has some code re-factorization that does not necessarily reproduce the same speed. But we hope this code release could still benefit the community, and we are also looking forwarding to seeing future work making it blazingly fast. That said, there are many parameters in the config where you can tune. The current config is not the optimal for live running. Like I mentioned, there are settings like sync restriction, image resolution, etc that will also affect the speed a lot. **@monajalal** V100 GPU is also not a state-of-art GPU.

DATASETS

Name	Year	Content	Size	Other
ScanNet	2017-2018	RGB-D dataset with 2D and 3D data; collection of voxels	2.5 mio views in more than 1500 scans; annotated	ObjectFusion, SbO
NOCS	2019	RGB-D dataset of real indoor scenes + computer generated objects on real backgrounds	18 scenes, 6 object categories	BundleTrack
YCBInEOAT	2020	RGB-D real indoor dataset, with ground-truth poses; focus on occlusion and robot-manipulation	9 videos, 5 objects	BundleTrack

DATASETS

Name	Year			Other
ScanNet	2017-2018			ObjectFusion, SbO
NOCS	2019			BundleTrack
YCBInEOAT	2020	RGB-D real indoor manipulation	9 videos, 5 objects	BundleTrack


Made for 3D reconstruction of full indoor scenes


OK, but many object in the scene, better have smaller images with one object to start with


Lame, symmetric objects (bottle, can,…)
Same thing for HO3D dataset

DATASETS

Run on your custom data

- Prepare your RGBD video folder as below (also refer to the example milk data). You can find an example milk data here for testing.

```
root
  ├──rgb/    (PNG files)
  ├──depth/  (PNG files, stored in mm, uint16 format. Filename same as rgb)
  ├──masks/       (PNG files. Filename same as rgb. 0 is background. Else is foreground)
  └──cam_K.txt   (3x3 intrinsic matrix, use space and enter to delimit)
```

DATASETS

Run on your custom data

- Prepare your RGBD video folder as below (also refer to the example milk data). You can find an example milk data here for testing.

```
root
  ├──rgb/    (PNG files)
  ├──depth/  (PNG files, stored in mm, uint16 format. Filename same as rgb)
  ├──masks/      (PNG files. Filename same as rgb. 0 is background. Else is foreground)
  └──cam_K.txt   (3x3 intrinsic matrix, use space and enter to delimit)
```

Masks for each frame!?

DATASETS

Run on your custom data

- Prepare your RGBD video folder as below (also refer to the example milk data). You can find an example milk data here for testing.

```
root
  ├──rgb/    (PNG files)
  ├──depth/  (PNG files, stored in mm, uint16 format. Filename same as rgb)
  ├──masks/       (PNG files. Filename same as rgb. 0 is background. Else is foreground)
  └──cam_K.txt   (3x3 intrinsic matrix, use space and enter to delimit)
```

?

Open | cam_K.txt ~/Desktop/ETH/Bachelor/Datasets/2022-11-18-15-10-24_milk | Save

```
1 6.153791503906250000e+02 0.000000000000000000e+00 3.147796325683593750e+02
2 0.000000000000000000e+00 6.153792114257812500e+02 2.366492004394531250e+02
3 0.000000000000000000e+00 0.000000000000000000e+00 1.000000000000000000e+00
```

BundleTrack (2021)

- Keep working on this code
  - Try to modify it as possible to not make it run out of memory
    - Resize images
    - Change batch size
    - Tune other config settings
- Prepare my scenes
  - Record one simple object, if everything is fine move to more complicated ones
- Try again running BundleTrack