Supplementary for MultiEgo

1 MultiEgo Dataset Instruction

The dataset contains 5 scenes: talking, statement, concert, sword, and presentation. Each scene provide video, camera intrinsic, camera poses, timestamp, and a sparse point cloud of the first frame scene.

The file construction is as follows:

```
scene
|-cam1
| |-<scene>-cam1.mp4
| |-intrinsic.txt
| |-camera_poses.txt
| |-sampletime.txt
|-cam2
|-cam3
|-cam4
|-cam5
|-sparse
|-camera.bin
|-images.bin
|-points3D.bin
|-points3D.ply
```

where `<scene>-camx.mp4` is the egocentric video of the performer x in the scene. If frame extraction is performed on all videos, it is recommended to reserve 25 GB of storage space.

`intrinsic.txt` is the intrinsic matrix of the camera x, in the format as:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

`camera_poses.txt` is the camera poses matrix of the frames in the `<scene>-camx.mp4`. The camera poses are represented as camera-to-world transformations in the world coordinate system. The pose in the format as:

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{2}$$

`sampletime.txt` is the capture time of the acquisition system. The data in `sampletime.txt` is in the unit of nano-second.

The `sparse` directory contains COLMAP [2] binary files for all images, including intrinsic camera parameters (`camera.bin`) and world-to-camera extrinsic transformations (`images.bin`).

The `images.bin` file names follow the naming convention `camx_frame_00000.png`. Additionally, we provide sparse 3D point clouds reconstructed from the first frame's images and extensive images, stored in `points3D.bin` and `points3D.ply`.

2 Data Loader Example

a data loading pipeline example: Modified from `dataset_readers.py` in 4DGaussian [5].

```python
#
# Copyright (C) 2023, Inria
# GRAPHDECO research group,
    https://team.inria.fr/graphdeco
# All rights reserved.
#
# This software is free for non-commercial, research
    and evaluation use
# under the terms of the LICENSE.md file.
#
# For inquiries contact george.drettakis@inria.fr
#

import os
...

class CameraInfo(NamedTuple):
...

class SceneInfo(NamedTuple):
...

def getNerfppNorm(cam_info):
...


def getTimeScale(scene):
timescale=[]
for num in range(1,6):
with
    open(f'path/to/{scene}/cam{num}/sampletime.txt','r')
    as f:
txt=f.readlines()
time=[]
for i in range(1,len(txt)): # the first line is an
    annotation
time.append(float(txt[i]))
timescale.append(time)
timescale=np.array(timescale)
timescale/=np.max(timescale)
return timescale

def getImageFolder(scene,cid,iid):
return f'/path/to/video/frames/{scene}
      /cam{cid}/frame_{iid:05d}.png'
```

```python
# In this example, the frames is in
    {scene}/cam{x}/frame_00000.png

def getCandIid(name): # get cam and image id from
    image_name in colmap .bin file
split=name.split('_')
cid=int(split[0][-1])
iid=int(split[2])
return cid,iid

def readColmapCameras(scene,cam_extrinsics,
    cam_intrinsics, images_folder):
# Read the entire timeline before the loop
timescale_all=getTimeScale(scene)

cam_infos = []
for idx, key in enumerate(cam_extrinsics):

sys.stdout.write('\r')
sys.stdout.write("Reading camera
    {}/{}".format(idx+1, len(cam_extrinsics)))
sys.stdout.flush()

# scale the camera and image
scale=0.5

extr = cam_extrinsics[key]
intr = cam_intrinsics[extr.camera_id]
height = int(intr.height*scale)
width = int(intr.width*scale)


uid = intr.id
R = np.transpose(qvec2rotmat(extr.qvec))
T = np.array(extr.tvec)

if intr.model in ["SIMPLE_PINHOLE", "SIMPLE_RADIAL"]:
focal_length_x = intr.params[0]*scale
FovY = focal2fov(focal_length_x, height)
FovX = focal2fov(focal_length_x, width)
elif intr.model=="PINHOLE":
focal_length_x = intr.params[0]*scale
focal_length_y = intr.params[1]*scale
FovY = focal2fov(focal_length_y, height)
FovX = focal2fov(focal_length_x, width)
elif intr.model == "OPENCV":
focal_length_x = intr.params[0]*scale
focal_length_y = intr.params[1]*scale
FovY = focal2fov(focal_length_y, height)
FovX = focal2fov(focal_length_x, width)
else:
assert False, "Colmap camera model not handled: only
    undistorted datasets (PINHOLE or SIMPLE_PINHOLE
    cameras) supported!"

# get cam num and frame id from image_name
cam_num,img_id=getCandIid(os.path.basename(extr.name).
       split(".")[0])
# get image path
image_path = getImageFolder(scene,cam_num,img_id) #
    os.path.join(images_folder,
    os.path.basename(extr.name))
image_name =
    os.path.basename(image_path).split(".")[0]
image = Image.open(image_path).resize((width,height))
image = PILtoTorch(image,None)
# get timestamp (or automatic allocation)
time= timescale_all[cam_num-1,img_id-1] #
    float(img_id/len(timescale_all[0]))

cam_info = CameraInfo(uid=uid, R=R, T=T, FovY=FovY,
    FovX=FovX,
    image=image,camera_id=cam_num,image_path=image_path,
    image_name=image_name, width=width,
    height=height,time = time, mask=None)
cam_infos.append(cam_info)
sys.stdout.write('\n')
return cam_infos

def fetchPly(path):
plydata = PlyData.read(path)
vertices = plydata['vertex']
positions = np.vstack([vertices['x'], vertices['y'],
    vertices['z']]).T
colors = np.vstack([vertices['red'],
    vertices['green'], vertices['blue']]).T / 255.0
# no such normals
normals = np.vstack([0, 0, 0]).T
return BasicPointCloud(points=positions,
    colors=colors, normals=normals)

def storePly(path, xyz, rgb):
...

# the boundaries of different scene
bound={'talking':[[-15,-5,-20],[25, 7, 14]],
       'statement':[[-15,-8,-25],[12, 6, 11]],
       'concert':[[-12,-15,-17],[15,7,12]],
       'sword':[[-10,-16,-5],[16, 5, 20]],
       'presentation':[[-10,-6,-3],[8, 5, 12]]}

# generate random point cloud
def randomPCD(scene):
```

```
num=1e5
xyz_scale=bound[scene]
x=np.random.uniform(xyz_scale[0][0],xyz_scale[1][0],num)
y=np.random.uniform(xyz_scale[0][1],xyz_scale[1][1],num)
z=np.random.uniform(xyz_scale[0][2],xyz_scale[1][2],num)
colors = np.random.randint(0, 256, size=(num, 3))
normals = np.zeros((num, 3))
xyz=np.array([x,y,z]).T

return BasicPointCloud(points=xyz, colors=colors,
    normals=normals)


def readColmapSceneInfo(path, images, eval,
    llffhold=8):
# get scene
scene=path.split('/')[-1]

try:
cameras_extrinsic_file = os.path.join(path,
    "sparse/0", "images.bin")
cameras_intrinsic_file = os.path.join(path,
    "sparse/0", "cameras.bin")
cam_extrinsics =
    read_extrinsics_binary(cameras_extrinsic_file)
cam_intrinsics =
    read_intrinsics_binary(cameras_intrinsic_file)
except:
cameras_extrinsic_file = os.path.join(path,
    "sparse/0", "images.txt")
cameras_intrinsic_file = os.path.join(path,
    "sparse/0", "cameras.txt")
cam_extrinsics =
    read_extrinsics_text(cameras_extrinsic_file)
cam_intrinsics =
    read_intrinsics_text(cameras_intrinsic_file)

reading_dir = "images" if images == None else images
cam_infos_unsorted = readColmapCameras(scene,
       cam_extrinsics=cam_extrinsics,
           cam_intrinsics=cam_intrinsics,
           images_folder=os.path.join(path,
           reading_dir))
cam_infos = sorted(cam_infos_unsorted.copy(), key =
    lambda x : x.image_name)

if eval:
train_cam_infos = [c for idx, c in
    enumerate(cam_infos) if idx % llffhold != 0]
test_cam_infos = [c for idx, c in
    enumerate(cam_infos) if idx % llffhold == 0]
else:
train_cam_infos = cam_infos
test_cam_infos = []

nerf_normalization = getNerfppNorm(train_cam_infos)

ply_path =
    f"/path/to/random/pointcloud/{scene}/randomply.ply"

bin_path = os.path.join(path,
    "sparse/0/points3D.bin")
txt_path = os.path.join(path,
    "sparse/0/points3D.txt")
if not os.path.exists(ply_path):
print("Converting point3d.bin to .ply, will happen
    only the first time you open the scene.")
try:
xyz, rgb, _ = read_points3D_binary(bin_path)
except:
xyz, rgb, _ = read_points3D_text(txt_path)
storePly(ply_path, xyz, rgb)

## choose one
# pcd=randomPCD()
pcd = fetchPly(ply_path)


scene_info = SceneInfo(point_cloud=pcd,
train_cameras=train_cam_infos,
test_cameras=test_cam_infos,
video_cameras=train_cam_infos,
maxtime=0,
nerf_normalization=nerf_normalization,
ply_path=ply_path)
return scene_info
def generateCamerasFromTransforms(path,
    template_transformsfile, extension, maxtime):
...
### no changes followed
```

3 Data Processing Details

In the following part, we will explain the details of data annotation process. We assume that after a data acquisition, the i-th AR glasses acquires a sequence of image frames X_i, and a sequence of gyroscopic pose frames G_i.

3.1 Monocular Pose Tracking

As described in Section 3.2 in the paper, each image frame and gyroscopic pose frame has its own timestamp, with image frames captured at 30Hz and gyroscopic pose frames at 50Hz. To align these data streams, we perform Spherical Linear Interpolation (SLERP) on the gyroscopic pose frames to obtain rotation data $\hat{G}_i$ corresponding to the exact capture times of the image frames. Specifically, let q_0 and q_1 denote the quaternions at times t_0 and t_1, respectively. The interpolated quaternion q at time $t \in (t_0, t_1)$ is given by:

$$q = q_0 (q_0^{-1} q_1)^{\frac{t - t_0}{t_1 - t_0}} \tag{3}$$

Then we employ several different image-based camera pose estimation methods to obtain multiple camera trajectories, in this paper we use Anycam [4], Mega-SAM [6], CUT3R [3], MonST3R [6] and PySLAM [1]. We let $P_{i,j}$ denote the j-th trajectory of i-th image frame sequence X_i, where the translation part is $t_{i,j}$ and the rotation part is $r_{i,j}$. It's notable that we . Subsequently, we fuse all the trajectories based on the rotation data q obtained by SLERP. Specifically, we calculate the importance m_j of j-th method based on the L_1 norms of the difference between $\hat{G}_i$ and $r_{i,j}$:

$$m_j = \frac{1}{\sum_i^I |r_{i,j}^{-1} \hat{G}_i| / I} \tag{4}$$

where I is the number of AR glasses. We obtain the weight w_j of the j-th method based on m_j:

$$w_j = \frac{m_j}{\sum_J^n m_n} \tag{5}$$

After the calculation above, a normalized monocular camera trajectory $\bar{P}_i$ of the i-th AR glasses is given by:

$$\bar{P}_i = \left(\sum_j^J w_j \cdot \frac{t_{i,j}}{\|t_{i,j,max}\|}, \frac{\sum_j^J w_j \cdot r_{i,j}}{\|\sum_j^J w_j \cdot r_{i,j}\|}\right) \tag{6}$$

where $\sum_j^J w_j \cdot \frac{t_{i,j}}{\|t_{i,j,max}\|}$ denotes the translation part, and $\frac{\sum_j^J w_j \cdot r_{i,j}}{\|\sum_j^J w_j \cdot r_{i,j}\|}$ denotes the rotation part. We abbreviate them as $\bar{t}_i$ and $\bar{r}_i$, respectively.

3.2 Multi-camera Pose Synthesis

Before data acquisition, we capture supplementary image sequence X_s of the first frame static scene. We process the supplementary image sequence X_s and the first frame of all the image frame sequence X_i by SfM pipeline of COLMAP [2] to reconstruction a static scene. In this scene, we obtain the absolute pose of different AR glasses at first frame $P_{i,0}$. Then we add the images in X_i which have the max translation value, into the static scene to obtain the absolute pose of these images. We denote the displacement value between the first frame pose and the corresponding max translation pose as $\Delta t_{i,max}$. To scaling the normalized monocular trajectory $\bar{P}_i$ to the size of the static scene, we calculate a scale factor s_i:

$$s_i = \frac{\|\Delta t_{i,max}\|}{\|\bar{t}_{i,max}\|} \tag{7}$$

Then, based on normalized monocular pose $\bar{P}_i$ and scale factor s_i, the absolute pose sequence of i-th view P_i is given by:

$$P_i = (t_{i,0} + s_i \cdot \bar{t}_i \cdot r_{i,0}, \quad r_{i,0} \cdot \bar{r}_i) \tag{8}$$

where $t_{i,0}$ and $r_{i,0}$ denotes the translation and rotation of first frame pose, $t_{i,0} + s_i \cdot \bar{t}_i \cdot r_{i,0}$ and $r_{i,0} \cdot \bar{r}_i$ represent the translation and rotation of the final absolute pose.

4 Consent Forms

Consent forms of performers are shown in figure 1.

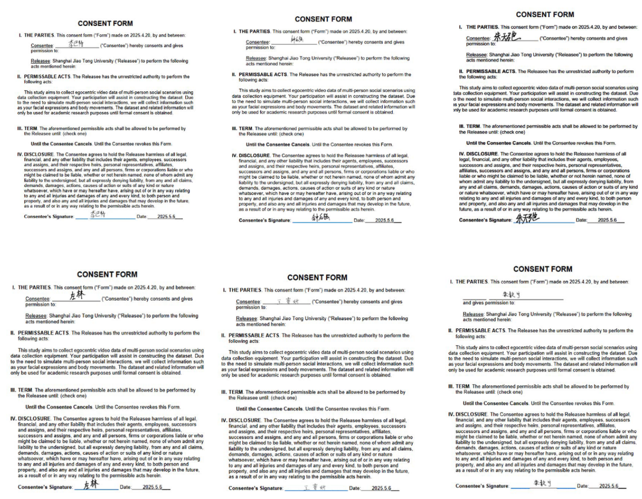

Figure 1: Consent Forms of Performers

References

[1] Luigi Freda. 2025. pySLAM: An open-source, modular, and extensible framework for SLAM. *arXiv preprint arXiv:2502.11955* (2025).

[2] Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[3] Qianqian Wang*, Yifei Zhang*, Aleksander Holynski, Alexei A. Efros, and Angjoo Kanazawa. 2025. Continuous 3D Perception Model with Persistent State. In *CVPR*.

[4] Felix Wimbauer, Weirong Chen, Dominik Muhle, Christian Rupprecht, and Daniel Cremers. 2025. AnyCam: Learning to Recover Camera Poses and Intrinsics from Casual Videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

[5] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 2024. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 20310–20320.

[6] Junyi Zhang, Charles Herrmann, Junhwa Hur, Varun Jampani, Trevor Darrell, Forrester Cole, Deqing Sun, and Ming-Hsuan Yang. 2024. MonST3R: A Simple Approach for Estimating Geometry in the Presence of Motion. *arXiv preprint arxiv:2410.03825* (2024).