INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 4
MPEG VIDEO CODING

ISO/IEC JTC 1/SC 29/WG 4 m66419
January 2024, Online

Title: **[INVR] Guideline of ReRF Extended Implementation**

Source: **Jun-Hyeong Park, Jaeyeol Choi, Jong-Beom Jeong (SKKU), Jinho Lee, Gun Bang (ETRI), Eun-Seok Ryu (SKKU)**

1 Introduction

During the last 144th meeting, two documents related to the residual radiance fields (ReRF)[1] model were presented on EE2 [2][3]. ReRF utilizes an explicit voxel grid format, learning both residual and motion information in an inter-mode strategy, efficiently modelling dynamic environments within Multiview video dataset.

The approach of ReRF, with its suitability for achieving streaming and temporal consistency, aligns well with the objectives of 3D INVR. Consequently, the output document of last meeting recommended ongoing exploration of ReRF within EE2 [4].

Although the official code for ReRF is provided by its authors, it lacks support for scene datasets, necessitating minor structural and code modifications for INVR experiments. To facilitate these experiments and enhance user convenience, an extended version of the ReRF code is being shared on INVR GitLab. (https://git.mpeg.expert/MPEG/Video/invr/softwares/rerf.git) This document aims to provide a simple guideline for using the currently available version of the ReRF code.

2 ReRF Model Enhancements

The original version of the ReRF model primarily focused on object-centric data reconstruction. Effective in this specific dataset, it was, however, limited in scenarios primarily involving discrete objects, thus not fully implemented to handle complex dynamic scenes with more intricate details.

The enhanced ReRF model incorporates methods from "Improved Direct Voxel Grid Optimization for Radiance Fields Reconstruction."[5] This paper proposes a voxel-grid-based approach for modelling forward-facing scenes. Central to this enhancement is the adoption of points parameterization and sampling, paralleling the techniques used in neural radiance fields (NeRF). An important aspect of this development is the use of dense voxel grids, conceptually similar to multiplane images (MPIs). Through the strategic placement of multiple RGB-density images at fixed depths, each with a unique resolution, and sampling rays across these images, the model

achieves a more refined reconstruction of scenes. This approach improves depth perception and spatial complexity, facilitating the capture of detailed scene variations more effectively.

Consequently, the extended ReRF model demonstrates improved ability to reconstruct forward-facing scenes with natural content and higher complexity. A comparative figure illustrates the reconstruction results from both the original and enhanced versions of the ReRF model, highlighting the advancements made.

(a)

(b)

Figure 1. Comparison of rendering quality between original and extended ReRF models. (a) Rendered image of original ReRF (left: v11, right: v27), (b) Rendered image of extended ReRF (left: v11, right: v27)

Despite these improvements, experiments with various mandated sequences indicated the need for configuration adjustments based on the specific characteristics of each sequence. These adjustments and their implications will be discussed in Section 3.2.

3 ReRF Usage Guide

3.1 Preparing Dataset

The initial phase in utilizing the ReRF model involves the meticulous preparation of the dataset. The dataset preparation requires multi-view videos in MP4 format, paired with an N*17 'poses_bounds.npy' camera pose file, following the NV3D dataset's format for compatibility with the ReRF model. For the acquisition of these camera pose files, the use of Colmap is recommended

due to its accuracy and reliability in generating pose data. Subsequently, the video files must be segmented into individual frames, a procedure efficiently executed using the 'llff_data_util.py' script. This script additionally generates a bounding box (bbox) file, crucial for delineating the scope of the scene. It is imperative to adjust the parameters of this bbox file to align with the specific dimensions and characteristics of the sequence, thereby optimizing the conditions for model training.

Table 1. Code snippet for preparing llff dataset for ReRF training

```
# need to adjust it to the size of your dataset
def create_bbox(llff_dataset_path):
    bbox = {
        "xyz_min": [-1.5, -1.5, -1.0],
        "xyz_max": [1.5, 1.5, 1.0]
    }

    with open(os.path.join(llff_dataset_path, 'bbox.json'), 'w') as json_file:
        json.dump(bbox, json_file, indent=4)


def extract_number(filename):
    """Extracts the number from the filename."""
    match = re.search(r'(\d+)', filename)
    if match:
        return int(match.group(1))
    return 0

def convert_nv3d_to_llff(video_dataset_path, llff_dataset_path, num_frames):
    poses_bounds = np.load(os.path.join(video_dataset_path, 'poses_bounds.npy'))
    video_files = sorted(os.listdir(video_dataset_path), key=extract_number)
     video_files = [f for f in video_files if f.endswith('.mp4')]

    for frame_id in tqdm(range(num_frames), desc="Processing frames"):
        frame_dir = os.path.join(llff_dataset_path, str(frame_id))
        os.makedirs(frame_dir, exist_ok=True)
        image_dir = os.path.join(frame_dir, 'images')
        os.makedirs(image_dir, exist_ok=True)

        for view_id, video_file in enumerate(video_files):
            video = cv2.VideoCapture(os.path.join(video_dataset_path, video_file))
            video.set(cv2.CAP_PROP_POS_FRAMES, frame_id)
            current_frame_id = int(video.get(cv2.CAP_PROP_POS_FRAMES))
            if current_frame_id != frame_id:
                print(f"Warning: Skipping frame {frame_id} in video {video_file}.
Could not seek to the correct position.")
                continue

            success, image = video.read()

            if success:
                cv2.imwrite(os.path.join(image_dir,
f'image_{str(view_id).zfill(4)}.png'), image)

        np.save(os.path.join(frame_dir, 'poses_bounds.npy'), poses_bounds)

convert_nv3d_to_llff(video_path, llff_path, num_frames)
```

```
create_bbox(llff_path)
```

3.2 Training Model

Once the dataset is prepared, the subsequent step is initiating the training process. This phase requires the creation of configuration files, which need to be carefully tailored to the specific characteristics of each sequence. These configuration files are designed to be flexible, allowing for adjustments in various hyperparameters to optimize the performance of the model. Annotations within the codebase guide users on potential modifications to these hyperparameters, enabling fine-tuning of the model to meet specific requirements. A set of pre-configured files for INVR mandated sequences are readily available in the repository.

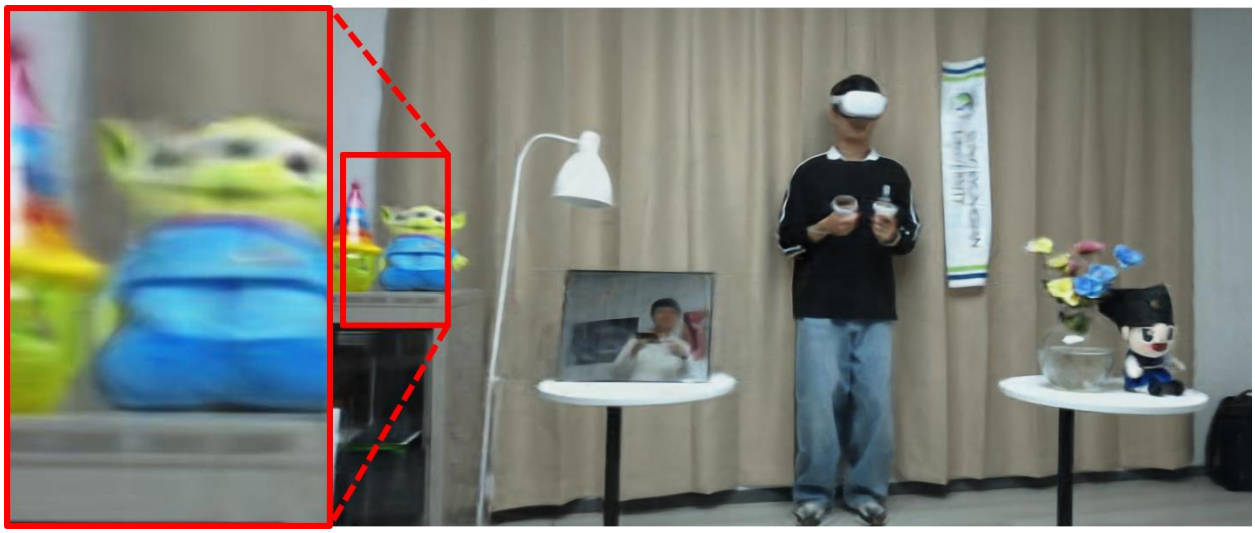
Table 2. Code snippet of configuration file for *SKKU_VRroom_1D* sequence

```
_base_ = '../default.py'

expname = 'skku_vr'
basedir = './output'

train_mode = 'sequential'
fix_rgbnet = True
frame_num = 32                                  # set frame_num to train
res_lambda=1e-2
deform_lambda=1e-2
deform_res_mode= 'separate'
use_res = False
use_deform = ''

data = dict(
    datadir= './data/SKKU',                     # path to dataset root folder
    dataset_type= 'llff',
    inverse_y=False,
    flip_x=False,
    flip_y=False,
    annot_path='',
    split_path='',
    sequence_name='',
    load2gpu_on_the_fly=True,                   # do not load all images into gpu
    testskip=1,                                 # subsample testset to results
    white_bkgd=False,                           # use white background
    half_res=False,
    ndc=True,                                   # use ndc coordinate
    spherify=False,                             # inward-facing
    factor=1,                                   # down-sampling
    llffhold=0,                                 # testsplit
    load_depths=False                           # load depth

)

deform_from_start=False

coarse_train = dict(
```

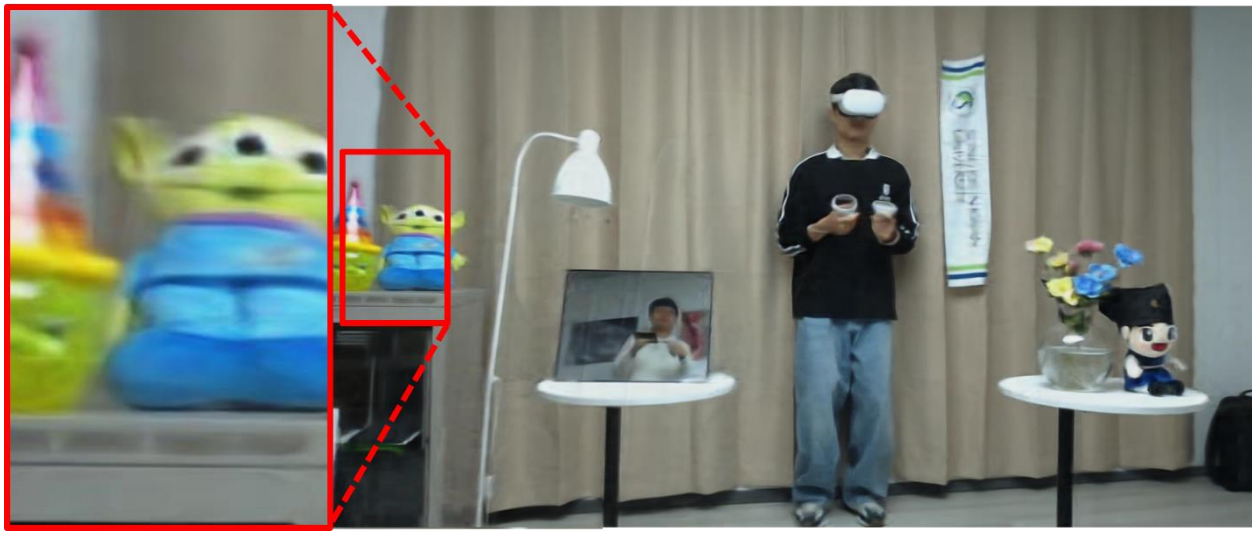
```
    N_iters=0,                                  # we don't need the coarse stage
training
)

fine_train=dict(
    N_iters=30000,
    N_rand=4096,                               # it seems that larger batch don't help
    weight_distortion=0.01,
    pg_scale=[],
    pg_scale_pretrained=[],
    decay_after_scale=0.1,
    ray_sampler='flatten',
    tv_before=1e9,                             # enable total variation loss
    tv_dense_before=10000,                     # dense version of total variation loss
                                               # for the first 10k iterations
    weight_tv_density=1e-5,
    weight_tv_k0=1e-6,
    lrate_deformation_field=1e-4

)

_mpi_depth = 384                              # the number of planes in Multiplane Image
_stepsize = 1.0

fine_model_and_render = dict(
    maskout_near_cam_vox=False,
    num_voxels=384*384*_mpi_depth,
    mpi_depth=_mpi_depth,
    stepsize=_stepsize,
    rgbnet_dim=9,                        # it seems that more rgbnet_dim don't help
    rgbnet_width=64,                    # it seems that larger rgbnet_width don't help
    world_bound_scale=1,                 # we don't have to slightly enlarge the ndc
    fast_color_thres=_stepsize/_mpi_depth/5,

)
```

To initiate and carry out the training, specific command-line instructions are required. Detailed commands and guidelines for executing the training process are documented in the README file available in repository.

Following figure illustrates the impact of configuration adjustments on model quality. It shows the results of enhancing the reconstruction quality through configuration tuning in the same sequence.



(a)



(b)

Figure 2. Comparison of high-frequency region reconstruction of v11 through hyperparameter tuning, (a) without hyperparameter tuning, (b) with hyperparameter tuning

3.3 Compression and Rendering

After training, the model can be compressed using the compression module implemented in ReRF, located in the codec folder under the compress directory. Users can modify these files for custom compression quality or process adjustments.

A modification in this version of ReRF is the ability to render using custom poses, easing the process of extracting test views in INVR experiments. This code supports two rendering methods: uncompressed and post-compression with decoding. The availability of uncompressed rendering is particularly crucial as it serves as a benchmark to evaluate the extent to which compression

affects the quality of scene reconstruction. By comparing results from both uncompressed and compressed rendering, we can assess the trade-offs between file size and reconstruction fidelity. For detailed instructions on each rendering method, users are encouraged to refer to the README file.

4 Conclusion

This document offers guidelines for using the ReRF model in research. It details the current version's improvements and performance. The code is continually being updated for enhanced functionality. We recommend researchers in INVR EE2 to use this evolving ReRF code in their further exploration. In case of any related requests, please contact one of the software coordinators:

- Gun Bang, gbang@etri.re.kr
- Jun-Hyeong Park, joke0702@skku.edu

5 References

[1] L. Wang, Q. Hu, Q. He, Z. Wang, J. Yu, T. Tuytelaars, L. Xu, M. Wu, “Neural Residual Radiance Fields for Streamably Free-Viewpoint Videos”, CVPR 2023, June, 2023.

[2] J. Lee, H. Lee, G. Bang, [INVR] EE2.2 ReRF investigation report for 3D INVR, ISO/IEC JTC 1/SC 29/WG 4/m65170, Hannover, Oct 2023.

[3] J. -H. Park, J. Choi, J. -B. Jeong, E. -S. Ryu, [INVR] Report on EE2.1: Inter Mode Dynamic NeRF Investigation, ISO/IEC JTC 1/SC 29/WG 4/m64722, Hannover, Oct 2023.

[4] G. Bang, Y. Liao, P. Hellier, M. Teratani, “Exploration experiments on implicit neural visual representation”, ISO/IEC JTC 1/SC 29/WG 04 N0426, Geneva, July 2023.

[5] C. Sun, M. Sun, H. -T. Chen, “Improved direct voxel grid optimization for radiance fields reconstruction”, arXiv preprint arXiv:2206.05085, 2022.