# URF: Urban Radiance Fields

## Supplementary Material

The following supplemental material contains additional implementation details, ablation studies, qualitative results, and a discussion of potential negative societal impacts that would not fit in the main paper.

## A. Additional Implementation Details

### A.1. Network architecture

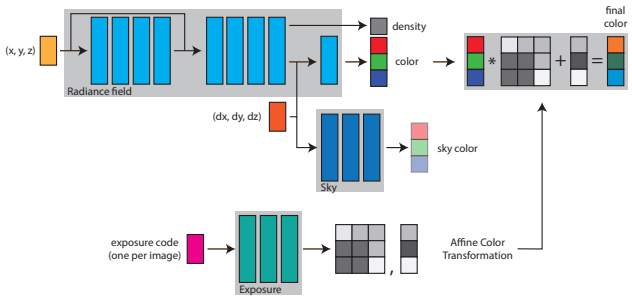



Figure 1. **Network architecture**

Our network architecture is illustrated in Fig. 1 and has three components. The first component is the neural radiance field network, which is designed similarly to the original network in NeRF [10]. It consists of a series of fully connected layers of width 256 that take as input the 3D location of a point $x, y, z$ and the viewing direction $dx, dy, dz$ and output the RGB color and the density at that point. The second component is the sky network, which takes as an input the direction $dx, dy, dz$ of a ray pointing at a sky point, and outputs its color. Finally, the third component is an exposure compensation network that takes as an input an exposure latent code and estimates the affine transformation to be applied to the color values output by the radiance field network. There is a different affine transformation per image. This compensates for the different exposures across input images. All three network are trained jointly so that the final colors output by the model will match the pixel colors in the input images.

### A.2. Training Protocol

We train a separate network for each baseline model (Sec. 5.2) and each variant of our model, applies to each scene. Every network is trained with the same protocol, detailed here. We use a TPU v2 architecture with 128 cores [5] using Tensorflow 2 [1]. We used the Adam optimizer [7] with a learning rate scheduler that included two stages. The warm up stage lasts 50 epochs, with the learning rate starting at 0.0005 and growing linearly until 0.005. After warm up, the main stage lasts 500 epochs, with the learning rate starting at 0.005 and then decaying exponentially with exponent 0.98. The ray batch size was set to 2048 per core and the total training time was about one day per network.

For ray sampling, we use a stratified strategy where the intervals are evenly spaced in log scale, and we sample 1024 samples per ray. We did not perform hierarchical sampling. Each batch contains rays randomly sampled from all images (and similarly for the lidar points)

The 3D location of a point is described using integrated positional encoding [3] with $L = 10$ frequencies. For the viewing direction we use the original positional encoding representation with 4 frequencies.

## B. Additional Ablation Studies

### B.1. Effect of margin $\epsilon$

| | Avg Error↓ | CD↓ | Acc↑ | F↑ |
|---|---|---|---|---|
| Fixed | 1.007 | 2.195 | 0.814 | 0.871 |
| Stepwise | 0.776 | 1.818 | 0.849 | 0.905 |
| Linear | 0.238 | 0.508 | 0.903 | 0.961 |
| Exponential | 0.249 | 0.863 | 0.901 | 0.966 |

Table 1. **Margin decay ($\epsilon$)** – We evaluate different decay strategies for the margin $\epsilon$ during training in the Rome scene. The margin controls the contribution of the lidar losses $\mathcal{L}_{\text{near}}$ and $\mathcal{L}_{\text{empty}}$. Having a fixed margin results in lower performance, while gradually decreasing it performs the best.

As we observed in Sec. 5.5 of the main paper, the empty-space loss can actually decrease 3D reconstruction performance as it introduces a strong preference for empty space. Using the near-surface loss, which is complementary to empty-space by construction, alleviates this effect. In Tab. 1 we vary the margin $\epsilon$ in Eq. (16) and Eq. (17) during training using different strategies: Fixed: keep a constant margin throughout training (as in [2]); Stepwise: start with a large margin (thus only the near-surface loss is activated) and after $N = 50$ epochs the margin suddenly becomes small; Linear/exponential: gradually reduce the margin from large to small with a linear or an exponential schedule. For all methods, the smallest value $\epsilon$ was set to 20cm. The linear and exponential strategies perform similarly and much better than the fixed and stepwise ones, indicating that the empty-space loss is best applied after the training process manages to infer a good initial version of the scene structure.

### B.2. Effect of exposure handling

In Tab. 2 we expand the ablation experiment in Sec. 5.5 of the main paper, comparing our affine transformation model

| | $D$ | Avg Error↓ | CD↓ | Acc↑ | F↑ |
|---|---|---|---|---|---|
| Direct | 48 | 1.071 | 1.28 | 0.159 | 0.362 |
| Affine | | 0.98 | 1.007 | 0.253 | 0.47 |
| Direct | 4 | 1.049 | 2.062 | 0.247 | 0.477 |
| Affine | | 0.885 | 1.564 | 0.262 | 0.524 |

Table 2. **Exposure handling** – We compare our affine transformation model with the direct input of the exposure code to the network. Using an explicit color transformation for the different exposures results in better reconstruction.

versus directly providing the exposure latent code to the network for the Rome scene. We experiment with two different dimensions for the latent codes, $D = 48$ as in NeRF-W [9] and a much smaller one $D = 4$. As Tab. 2 shows, our affine transformation approach performs better in all 3D reconstruction metrics, for both values of $D$. We also observe that both the affine and the direct approach perform better when the exposure latent code has smaller dimensionality, thus reduced capacity. For the direct approach this is in accordance to the observations in NeRF++ [12] about the viewing direction: implicit regularization (limiting the capacity) of the latent code can increase the performance. For the affine case, this indicates that there exist a compact latent space that can describe the color transformations appearing in the dataset and it is easier to learn.

### B.3. Tanks and Temples dataset

In order to demonstrate that our method does not only work on the Street View dataset, we ran it on *Tanks and Temples* [8]. Tab. 3 below shows that 3D reconstruction performance continuously improves while gradually adding our proposed components. In the "+SfM" column instead of lidar supervision, we run SfM using COLMAP and utilize the estimated 3D keypoints to supervise our losses from Sec. 4.2. A held-out set of 3D keypoints was used as ground truth for evaluation.

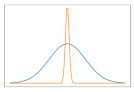
| | MipNeRF (base) | +exposure, +sky | +SfM (full) |
|---|---|---|---|
| Avg Err ↓ | 1.260 | 1.082 | 0.966 |
| Acc ↑ | 0.735 | 0.780 | 0.805 |

Table 3. **3D Reconstruction** – We compare variations of our model with the baseline (MipNeRF [3]) for the task of 3D reconstruction.

In terms of novel view synthesis (Fig. 2), our results on this dataset are state-of-the-art. We obtain 26.17 PNSR vs 22.37 for NeRF++[67], one of the best-performing previous methods, and similarly for LPIPS (0.246 vs 0.391, lower is better).


Figure 2. **Novel view synthesis and 3D reconstruction –** We visualize a novel view and the corresponding 3D reconstruction for the Playground scene from Tanks And Temples [8].

### B.4. Additional discussions

DS-NeRF [6] originally inputs depth cues from SfM. In our experiments, we modify it to input lidar instead. We do so using their expected depth supervision loss, as opposed to our three losses (Sec. 4.2). However, the expected depth loss does not constrain the distribution of weights along the ray, e.g. be peaky close to the surface.  Since this loss focuses on just a single value, the density along a ray can be very broadly distributed without penalization (see inset for two distributions with the same expected value). Moreover, DS-NeRF does not handle varying exposures, nor does it take care of the sky.

Additionally, we found that NeRF-W [9] image-dependent latent code is overfitting image-independent effects by inserting mid-density clouds along the camera line of sight. This results in a suboptimal 3D representation that makes inferring geometry difficult for held-out views. In the Held-out Building setting (see Sec. 5.1 for details), this overfitting issue is less severe as more of the scene is observed.

## C. Additional Qualitative Results

In Fig. 3 and 4 we show more visualizations of extracted colored meshes for different scenes. The color of every vertex is estimated by querying the radiance field network in that particular location. This representation can be used in common 3D editing software such as Blender [4] (first and second column in Fig. 4) and it allows for real time rendering on the browser, *e.g.* using ThreeJS [11] (third column in Fig. 4). Note that this way of rendering is different than the volumetric rendering in NeRF models, which is continuous and incorporates implicitly the view dependent appearance changes. Finally, we present additional results for novel view synthesis in Fig. 5.

## D. Potential Negative Societal Impacts

This section discusses potential negative impacts of our work on society.

**Privacy**. Experiments in this paper are run with data from a Street View dataset that contains image and lidar data


Figure 3. **Colored meshes –** We visualize the colored meshes estimated by our method for several Tanks And Temples [8] scenes.

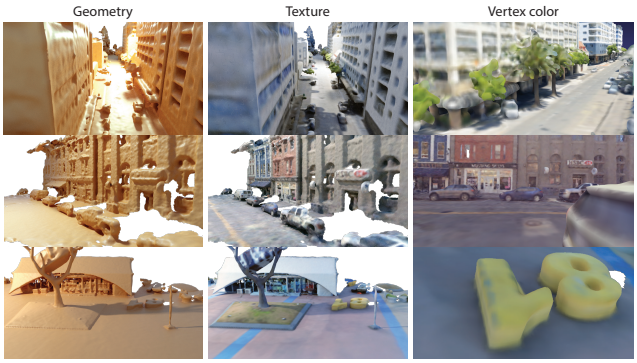



Figure 4. **Rendering colored meshes –** After extracting a color mesh using our model, we can render its geometry and texture in a 3D environment, or render the vertex colors in real time on a browser.

captured in public outdoor spaces. Of course, datasets of this type could potentially impact personal privacy. To mitigate this issue, we work only with images where people and license plates have been blurred and masked, we follow strict guidelines regarding how the data must be stored securely, and we do not redistribute any data, all in accordance with the privacy policy mandated by the data provider (https://policies.google.com/privacy).

**Fairness**. Experiments in this paper were run on data selected from twelve cities (Seattle, São Paulo, Toledo, Cape Town, Tokyo, Zurich, Rome, St Johns, Taipei, Sydney, Melbourne, and New York). Those cities were selected to provide a sampling of regions from around the world (six continents are represented) and to provide challenging test cases to stress the proposed algorithms. However, the sampling is strongly biased towards urban areas in major cities in this early study. This bias will be mitigated somewhat in future studies when the algorithms are run on all available data.

**Fakes**. This paper introduces a system to synthesize images, and thus could be used to create images with misleading or fake content, either intentionally or not. For example, the system might be better at reproducing imagery for certain types of shapes, materials, or lighting, leading to misleading novel images with systematic inaccuracies. Or, an adversary might be able to give the system images that are not from the same scene and/or do not match the lidar in order to achieve novel image results that do not match the real world. To mitigate these risks somewhat, we expect the system to be deployed only in controlled settings, where the input images are vetted and stored securely, and a sampling of the output images are checked manually for quality control. Additionally, we will clearly identify computer-generated images as synthetic.

**Energy consumption**. The paper proposes a method to train a coordinate neural network to produce novel images for a scene using volume rendering. As in any system of this type (e.g., NeRF), the optimization procedure to train the network is quite compute intensive, and thus has a negative impact on the environment due to its energy consumption. However, using lidar data as proposed in the paper helps the optimization converge more quickly, which saves some energy. Yet still the energy usage is very high, and further optimizations are warranted before any methods based on volume rendering with coordinate neural networks are ready to deploy at scale.

## References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 1

[2] Dejan Azinović, Ricardo Martin-Brualla, Dan B Goldman, Matthias Nießner, and Justus Thies. Neural rgb-d surface reconstruction. *arXiv*, 2021. 1

[3] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021. 1, 2

[4] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 2

[5] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc

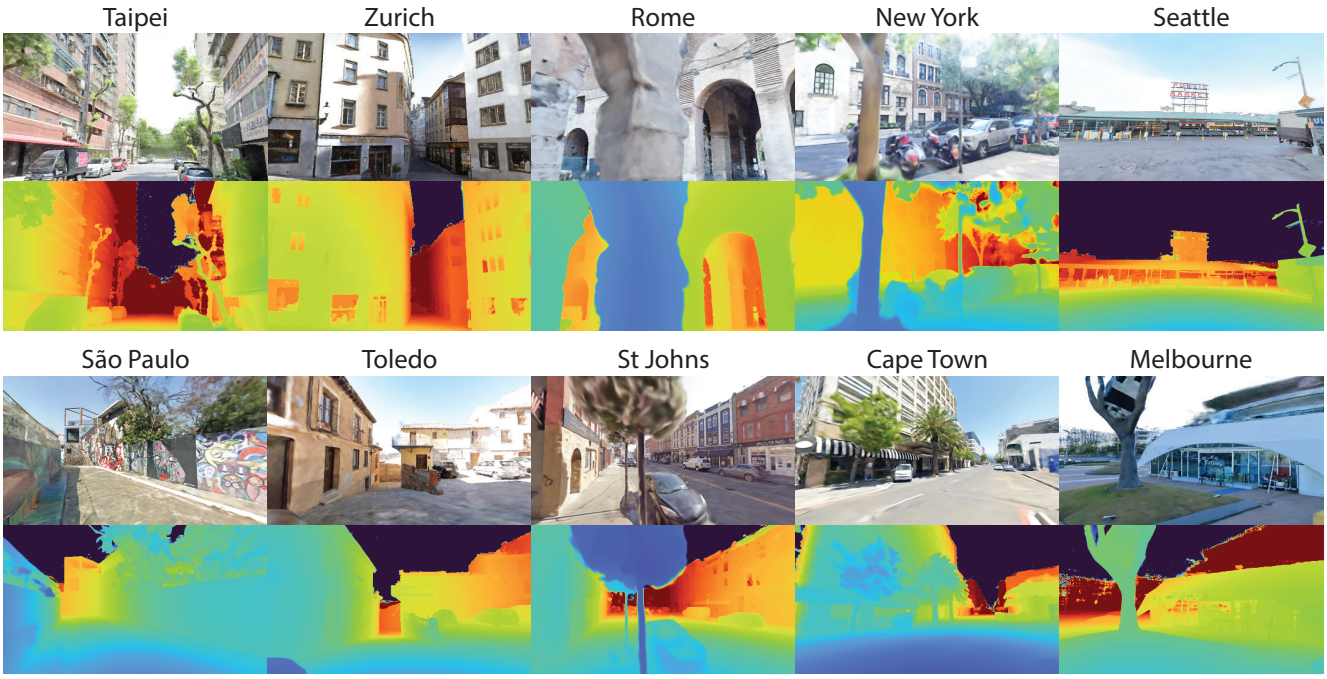



Figure 5. **Novel views**

Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2), 2017. 1

[6] Jun-Yan Zhu Kangle Deng, Andrew Liu and Deva Ramanan. Depth-supervised nerf: Fewer views and faster training for free. *arXiv:2107.02791*, 2021. 2

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 1

[8] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM TOG*, 2017. 2, 3

[9] Ricardo Martin-Brualla, Noha Radwan, Mehdi Sajjadi, Jonathan Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR*, 2021. 2

[10] Ben Mildenhall, Pratul Srinivasan, Matthew Tancik, Jonathan Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, pages 405–421. Springer, 2020. 1

[11] three.js. three.js, 2015. 2

[12] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020. 2