Pacific Graphics 2024
R. Chen, T. Ritschel, and E. Whiting
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 7

Supplementary Material: High-Quality Geometry and Texture Editing of Neural Radiance Field

Soongjin Kim[1] Jooeun Son[1] Gwangjin Ju[1] Joo Ho Lee[2] Seungyong Lee[†1]

[1]POSTECH, Korea
[2]Sogang University, Korea

1. Full Table of Quantitative Comparison

We provide the full quantitative results on each item in the DTU and NeRF synthetic datasets in Table 1 and Table 2, respectively.

Table 1: *Quantitative comparison on DTU datasets.*

Data	Metric	NeRF	NeuMesh	Ours
DTU24	PSNR ↑	23.786835	26.949687	**28.27720**
	LPIPS ↓	0.437067	0.281248	**0.170249**
	SSIM ↑	0.836081	0.929794	**0.963482**
DTU37	PSNR ↑	25.570391	25.767172	**27.20381**
	LPIPS ↓	0.27154	0.207288	**0.139661**
	SSIM ↑	0.930462	0.944740	**0.965532**
DTU40	PSNR ↑	26.676590	27.151396	**28.61123**
	LPIPS ↓	0.341483	0.318481	**0.213783**
	SSIM ↑	0.906628	0.932075	**0.961167**
DTU55	PSNR ↑	25.677692	27.655735	**28.16371**
	LPIPS ↓	0.267360	0.130560	**0.099138**
	SSIM ↑	0.935189	0.975775	**0.978725**
DTU97	PSNR ↑	25.65855	26.027949	**26.58995**
	LPIPS ↓	0.267292	0.186742	**0.137386**
	SSIM ↑	0.940272	0.952493	**0.961936**
DTU105	PSNR ↑	27.661314	28.165280	**28.31810**
	LPIPS ↓	0.268487	0.164451	**0.122389**
	SSIM ↑	0.958451	0.973944	**0.977826**
DTU110	PSNR ↑	25.318144	26.595053	**26.86536**
	LPIPS ↓	0.209471	0.157033	**0.128824**
	SSIM ↑	0.958383	**0.979175**	0.977116

2. Hyperparameters and Training Details

For training the radiance field in the surface-aligned volume, we employ the same hyperparameters as in the original TensoRF implementation. The height range was initialized to approximately $[-0.01, 0.01]$ for the DTU dataset and $[-0.03, 0.03]$ for the NeRF Synthetic dataset. Reconstructing a guide mesh takes about 5 minutes on a single NVIDIA GeForce RTX 3090 GPU. Training our model with the TensoRF backbone takes approximately 90 minutes on average, utilizing two NVIDIA GeForce RTX 3090 GPUs.

Table 2: *Quantitative comparison on NeRF Synthetic datasets. The NeuMesh paper does not provide the values on each item but only average values over all items.*

Data	Metric	NeRF	NeuMesh	Ours
chair	PSNR ↑	33.00	-	**35.89**
	LPIPS ↓	0.046	-	**0.025**
	SSIM ↑	0.967	-	**0.985**
hotdog	PSNR ↑	36.18	-	**37.00**
	LPIPS ↓	0.121	-	**0.041**
	SSIM ↑	0.974	-	**0.981**
lego	PSNR ↑	32.54	-	**34.83**
	LPIPS ↓	0.050	-	**0.028**
	SSIM ↑	0.961	-	**0.975**
mic	PSNR ↑	32.91	-	**36.45**
	LPIPS ↓	0.028	-	**0.018**
	SSIM ↑	0.949	-	**0.990**

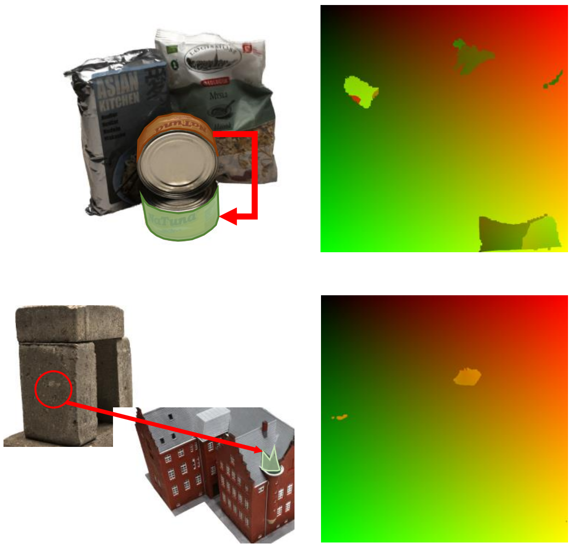
3. Detailed Algorithm and Pseudo Code

3.1. Scene initialization

We provide pseudo codes for computing the per-vertex height range (Algorithm 1) and creating the tetrahedron set (Algorithm 2) that are covered in Section 3.1 of the main paper.

3.2. Texture filling and swapping

For texture filling, we need to acquire a 2D mapping image that represents the *uv* changes from the source to the target appearance (Figure 1). When the source and target appearances are the same, the process is reduced to texture swapping. As the initialization step, we create a mapping image that represents the identity mapping for *uv* conversion. For texture editing, the user selects the source and target regions by marking faces in the guide meshes of the source and target scenes, respectively. Then, the two selected regions are aligned using Iterative-Closest-Point (ICP) operation to determine the correspondence between them. Finally, for the points in the source selected region, we find the closest points on the target



Operation Mapping Image

Figure 1: *Mapping image examples. We show mapping images for filling operation (top) and swapping operation (bottom). For visualization, the red and greed channels are used for representing u- and v-coordinates, respectively.*

surface and update the corresponding pixels in the mapping image using the target *uv*-coordinates. To prevent improper *uv*-locations from being involved at the region boundary, we use an additional binary mask to indicate where the *uv*-coordinates have been updated.

Algorithm 1 Per-vertex height range

Input: Vertex position v, Face ids f, Vertex normal vn, Initial range (u,d)

Output: Per-vertex upper range r_u, Per-vertex lower range r_d

function CLOSEST($v_1, n_1, v_2, n_2, u_t, d_t$)
  $V_{diff} \leftarrow v_2 - v_1$
  $N_{mat} \leftarrow [n_1, -n_2, n_1 \times n_2]$
  $T \leftarrow N_{mat}^{-1} \cdot V_{diff}$ ▷ Solve linear equation
  $t_1, t_2 \leftarrow T(0), T(1)$ ▷ Get corresponding solution
  return t_1, t_2
function SETHEIGHT($v, f, vn, (u,d)$)
  $N \leftarrow length(f)$
  $r_u \leftarrow length(v)$ sized array filled with u
  $r_d \leftarrow length(v)$ sized array filled with d
  for $k \leftarrow 1$ to N **do**
    $i_1, i_2, i_3 \leftarrow f(k)$
    $t_{10}, t_{20} \leftarrow CLOSEST(v(i_1), vn(i_1), v(i_2), vn(i_2))$
    $t_{11}, t_{30} \leftarrow CLOSEST(v(i_1), vn(i_1), v(i_3), vn(i_3))$
    $t_{21}, t_{31} \leftarrow CLOSEST(v(i_2), vn(i_2), v(i_3), vn(i_3))$
    for $(id, t_{val}) \leftarrow ((i_1, t_{10}), (i_1, t_{11}), (i_2, t_{20}), (i_2, t_{21}), (i_3, t_{30}), (i_3, t_{31}))$ **do**
      if $t_{val} > 0$ **then**
        $r_u(id) \leftarrow \min(t_{val}, r_u(id))$
      if $t_{val} < 0$ **then**
        $r_d(id) \leftarrow \max(t_{val}, r_d(id))$
  return r_u, r_d

Algorithm 2 Create Tetrahedron Set

Input: Vertex position v, Face ids f, Vertex normal vn, per-vertex upper range r_u, per-vertex lower range r_d

Output: Set of tetrahedrons

function TETVOLUME(v, f, vn, r_u, r_d)
  $f' \leftarrow sorted(f)$
  $N \leftarrow length(f')$
  $T \leftarrow []$
  for $k \leftarrow 1$ to N **do**
    $i_1, i_2, i_3 \leftarrow f'(k)$
    $v_a, v_b, v_c \leftarrow v(i_1), v(i_2), v(i_3)$
    $u_1 \leftarrow v_a + vn(i_1) * r_u(i_1)$
    $u_2 \leftarrow v_b + vn(i_2) * r_u(i_2)$
    $u_3 \leftarrow v_c + vn(i_3) * r_u(i_3)$
    $d_1 \leftarrow v_a + vn(i_1) * r_d(i_1)$
    $d_2 \leftarrow v_b + vn(i_2) * r_d(i_2)$
    $d_3 \leftarrow v_c + vn(i_3) * r_d(i_3)$
    $T.insert((u_1, u_2, u_3, d_1))$ ▷ Insert three tetrahedrons
    $T.insert((u_2, u_3, d_1, d_2))$
    $T.insert((u_3, d_1, d_2, d_3))$
  return T