SphereUFormer: A U-Shaped Transformer for Spherical 360 Perception

Supplementary Material

A. Technical Details

In this section, we describe the implementation of our method in detail.

A.1. Architecture

Given a group of nodes G_r of rank r on a sphere representation, an input data of rank r_0 is projected by an "Input Projection" layer, which is a fully connected layer mapping each node value to $D_1 = 32$ dimensions, then followed by GELU activation. In addition, an initial center downsampling layer reduces the rank of the data to $r_1 = r_0 - 1$. Global position encoding, which is explained in Appendix A.3, of dimension D_1 is added, forming the final embedded data.

$$e_i^0 = \text{GELU}(\text{Proj}_{in}(x_i)), \forall i \in G_{r_0} \quad (1)$$

$$\bar{e}^1 = \text{POOL}(e^0) \quad (2)$$

$$e_i^1 = \bar{e}_i^1 + \text{Pos}_0(\phi_i), \forall i \in G_{r_1} \quad (3)$$

The encoder consists of $N = 4$ consecutive spherical attention modules (SAM). Each consists of $M = 2$ spherical attention blocks (SAB). For each $n \in [1..4]$, the embedding dimension is $D_n = 2^{i-1} \cdot D_1$, resulting in 32, 64, 128, 256. At the end of each SAM is a downsampling block, which consists of a center downsampling layer, followed by a fully connected layer that projects the embedding from D_n to D_{n+1}. The downsampling reduces the sphere dimension from G_{r_0-n} to G_{r_0-n-1}. Therefore, for all $m, n \in [1..M] \times [1..N]$:

$$e^{n,m} = \text{SAB}_{n,m}(e^{n,m-1}) \quad (4)$$

$$e^{n+1} = \text{Proj}_n(\text{POOL}(e^{n,M})) \quad (5)$$

, where $e^{n,0}$ is e^n produced by the previous module.

As shown in Fig. 5 (in the main paper), each SAB is composed of a spherical local self attention (SLSA) with a residual connection, and an MLP that consists of 2 fully connected layers with in/out dimensions D_n and hidden dimension $4 \cdot D_n$, and a GELU activation.

$$\bar{h}^{n,m} = h^{n,m-1} + \text{SLSA}_{n,m}(h^{n,m-1}) \quad (6)$$

$$h^{n,m} = \bar{h}^{n,m} + \text{MLP}_{n,m}(\text{LayerNorm}(\bar{h}^{n,m})) \quad (7)$$

As can be seen in Fig. 4. The bottleneck follows the same structure as Eq. (4), operating on G_{r_0-N-1} with $D_{N+1} = 512$, and without a downsampling layer. The decoder follows the same structure as the encoder, with the slight modification that an upsampling with projection to D_n is performed instead of downsampling, and that the encodings are merged with a bypass connection from the encoder with the same sphere rank. For the decoder:

$$d^{n,0} = \text{UNPOOL}(\text{Proj}_n(d^{n+1})) \quad (8)$$

$$d^{n,0} = \text{CONCAT}(d^{n,0}, e^{n,M}) \quad (9)$$

$$d^{n,m} = \text{SAB}_{n,m}(d^{n,m-1}) \quad (10)$$

, where $d^n = d^{n,M}$. The concatenation also means that a decoder stage has a dimension of $2D_n$. Which is twice the size of the encoder.

Finally, a fully connected "output projection" projects the encoding d^1 from 64 channels to 1 for depth, and the number of classes in segmentation.

$$y = \text{Proj}_{out}(\text{UNPOOL}(d^1)) \quad (11)$$

A.2. Spherical Local Self Attention

The spherical local self attention (SLSA) Fig. 5 (in the main paper) is an attention layer that adheres to the sphere structure. A local operation is performed between neighboring nodes. A hyperparameter C_{win} controls how far through the graph neighbors are considered.

An input e with dimension D is projected into $\{q,k,v\}$ vectors. All have same the size, equal to $C_{head} \cdot D$. They are them split into H heads. Per stage, the encoder has 2,4,8,16 heads, the decoder has 4,8,16,16, and the bottleneck 16. Absolute position encoding is again added to q and k and then the vectors are normalized. Based on the graph structure G, for each e_i, we denote E_i^K as the set of neighbors given $C_{win} = K$. Then, for each node $i \in G$, the dot product between the query q_i and each key in its neighborhood. Relative position bias is added to each dot product result. A final softmax operation normalizes the attention weights. This is applied per attention head, but for conciseness, the head notation is not added.

$$\bar{a}_{i,j} = q_i \cdot k_j + \text{RelPos}(i,j), \forall i \in G, \forall j \in E_i^K \quad (12)$$

$$\{a_{i,1}, ..a_{i,|E_i^K|}\} = \text{Softmax}(\{\bar{a}_{i,1}, ..\bar{a}_{i,|E_i^K|}\}) \quad (13)$$

The attention map is then multiplied by the value vectors $v_j \forall j \in E_i^K$ to form the output of the head.

$$\bar{o}_i = a_{i,j} \cdot v_j, \forall i \in G, \forall j \in E_i^K \quad (14)$$

Finally, the different heads are concatenated and a linear output projection projects the concatenated heads to the input dimension.

$$o_i = \text{Proj}(\text{CONCAT}(\{\bar{o}_i^h\}_{h=1}^H)) \quad (15)$$

The encoder and decoder use 2,4,8,16 heads per equivalent stage $n \in [1..N]$.

A.3. Global Positional Encoding

Our method employs a vertical global positional encoding to inform the model of each node's vertical position on the sphere. The vertical position is provided by the angle $\phi \in [0, \pi]$, where 0 denotes the top point with $z = 1$ and π denotes the bottom point with $z = -1$.

To encode the position. The angle value ϕ is encoded with D sine and cosine wave functions of different frequencies, resulting in a total of $2D$ position features. The frequencies are sampled between $F_{min} = 1$ (fixed) and F_{max} (configurable) by the function:

$$f_i = F_{max}^{i/(D-1)}, \tag{16}$$

where i is a value between 0 and $D - 1$.

The position vector is then defined as:

$$\begin{aligned} \hat{\phi} &= (2\phi - \pi) \\ pos(\phi) &= [\sin(f_i\hat{\phi})|_{i=0}^{D-1}, \cos(f_i\hat{\phi})|_{i=0}^{D-1}] \end{aligned} \tag{17}$$

We used $D = 16$, $F_{max} = 10$. Fig. 9 illustrates the global position encoding.

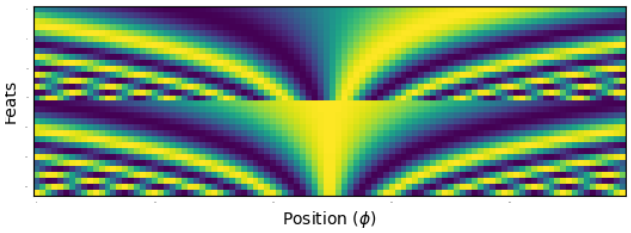

Figure 9. Visualizing the global position encoding per angle ϕ from 0 to π.

As we apply global position encoding in our model in different stages, we take the sinusoidal position encoding and project them with a learned linear layer to the dimension of the target vector, which can vary based on the stage within the architecture.

A.4. Relative Position Encoding

The relative position encoding has two steps. First, the relative angles $(\Delta\theta, \Delta\phi)$ are computed for each pair of neighboring nodes. Second, using the relative angles, a learned value is sampled from a 7×7 grid.

For a graph representation G, with K-order neighbor mapping E_i^K for each node $i \in G$. The angles $(\Delta\theta, \Delta\phi)$ are computed between each node i and each node $j \in E_i^K$. This is done by rotating the nodes based on θ_i, ϕ_i, and extracting the new $\Delta\theta_j, \Delta\phi_j$ after rotation. The rotated angles are computed by converting the nodes' positions to cartesian form (x, y, z), then multiplying by a rotation matrix, and converting back to spherical form.

$$x_j, y_j, z_j = \text{ToCartesian}(\theta_j, \phi_j) \tag{18}$$

$$[\hat{x}_j, \hat{y}_j, \hat{z}_j]^T = R_i * [x_j, y_j, z_j]^T \tag{19}$$

$$\Delta\theta_j, \Delta\phi_j = \text{ToSpherical}(\hat{x}_j, \hat{y}_j, \hat{z}_j) \tag{20}$$

The rotation matrix is defined by ϕ_i, θ_i as follows:

$$R_i = R_i^{\phi} * R_i^{\theta} \tag{21}$$

$$\hat{\theta}_i = \theta - \pi, \hat{\phi}_i = \phi - \frac{\pi}{2} \tag{22}$$

$$R_i^{\phi} = \begin{bmatrix} \cos(-\hat{\phi}_i) & 0 & \sin(-\hat{\phi}_i) \\ 0 & 1 & 0 \\ -\sin(-\hat{\phi}_i) & 0 & \cos(-\hat{\phi}_i) \end{bmatrix} \tag{23}$$

$$R_i^{\theta} = \begin{bmatrix} \cos(-\hat{\theta}_i) & -\sin(-\hat{\theta}_i) & 0 \\ \sin(-\hat{\theta}_i) & \cos(-\hat{\theta}_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{24}$$

After the rotation, node i receives $\Delta\theta_i = 0, \Delta\phi_i = 0$, while the neighbors on the right and left receive positive and negative (respectively) $\Delta\theta_j$ and above and below receive negative and positive (respectively) $\Delta\phi_j$. This computation is performed only once in advance, as the structure of the graph is fixed. After rotation, we use bilinear interpolation to sample the value in a 7×7 grid according to $\Delta\theta_j, \Delta\phi_j$. As was shown in Fig. 6 (in the main paper). Since the grid is in the range [-1,1] on both axes, we normalize $\Delta\theta_j, \Delta\phi_j$ by the max absolute value of each angle coordinate over all $i \in G$. The value is added as bias to the attention map.

B. Training Protocol

We trained each model for a similar amount of iterations. For Stanford2D3D [1] we used 25K iteration. For Structured3D [3], which is a much larger dataset, we trained for 160K iterations. Each iteration has a batch size 16. The models were trained with the Adam [2] optimizer, with a learning rate of 1e-4. Evaluation on the validation set was performed every 400 iterations, monitoring the best score.

B.1. Fast Resolution Upscaling

Since the model is composed of local operations. A model can be finetuned for higher resolution with pretrained weights trained on a lower resolution. As long as the hyperparameters of the network have not changed. We have found this to significantly speed up training, and convergence occurred in much fewer iterations. Since training on high resolution is much slower per iteration, this is a huge benefit. Fig. 10 shows the MRE throughout training on Stanford2D3D, for a rank 7 model, rank 8 model, and rank 8 model pretrained on rank 7 weights. As can be seen, when given enough iterations, rank 8 reaches rank 7's performance. But, a pretrained model on rank 7 converged much faster and already surpassed its counterparts in less than 5K iterations.

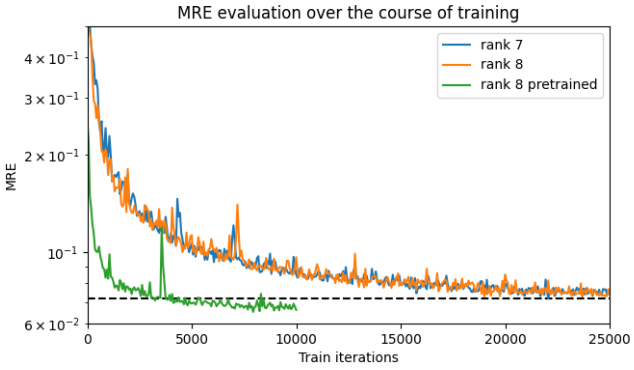

Figure 10. **High resolution training comparison.** Training a model in rank 7 and rank 8 resolutions converge at the same rate. However, fine-tuning a high resolution model on pretrained weights of a lower resolution converges much faster.

C. Performance Analysis

In this section, we performed a deeper analysis of our method's performance.

C.1. Boundary Effect

In our analysis of the baselines, we observed a recurring boundary effect at the points where the horizontal 360° and 0° meet. This boundary effect is most noticeable in depth estimation where the estimated depth is not continuous there. It can come as a misalignment between the two sides of the image, or as a vertical line that passes through the image. This boundary effect did not exist in our model, since it is fully horizontally equivariant. This is illustrated in Fig. 11, where we oriented the images to have the 0° at the center. It can be seen that all baselines suffer from this boundary issue to some extent, while our method does not.

C.2. Error Distribution on the Sphere

To better understand the strengths and weaknesses of each method. We also analyzed the error per location on the sphere. This is illustrated in Fig. 12, where we visualize the average error of many depth predictions over the validation set, normalized by the max error of all models. The visualization shows a low average error in red , and a high one in blue. As can be seen, our method has less error at the bottom and top (noticed by the darker red tone), due to the better handling of the distortions, while also having less error at the center (less bright spots), due to its higher effective resolution at the center. The higher effective resolution is a result of distributing the data points uniformly on the sphere, instead of the unbalanced sampling of ERP in favor of the top and bottom of the image.

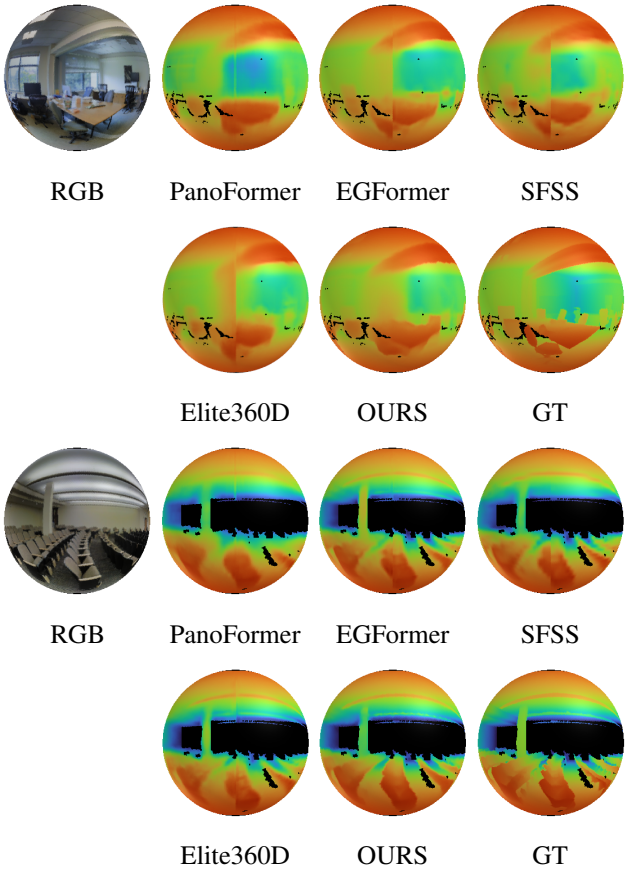

Figure 11. **Boundary effect comparison.** All baselines show a boundary issue at the 0° horizontal angle.

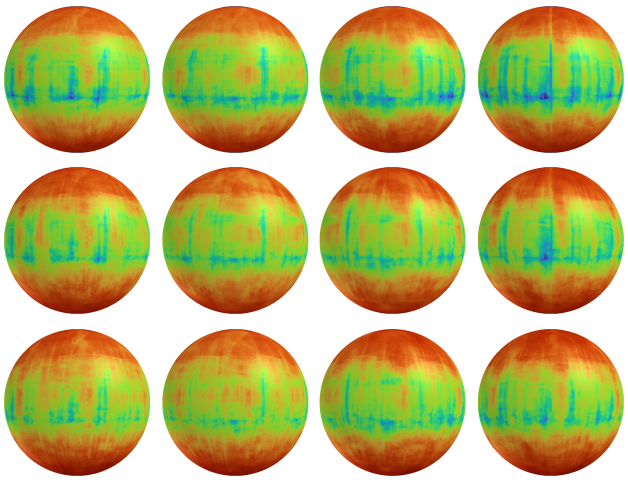

Figure 12. **MAE per location on the sphere. From top to bottom:** PanoFormer, EGFormer, Ours. **From left to right:** different sides of the 360° view. Dark red values indicate low error, while bright blue values indicate high error.

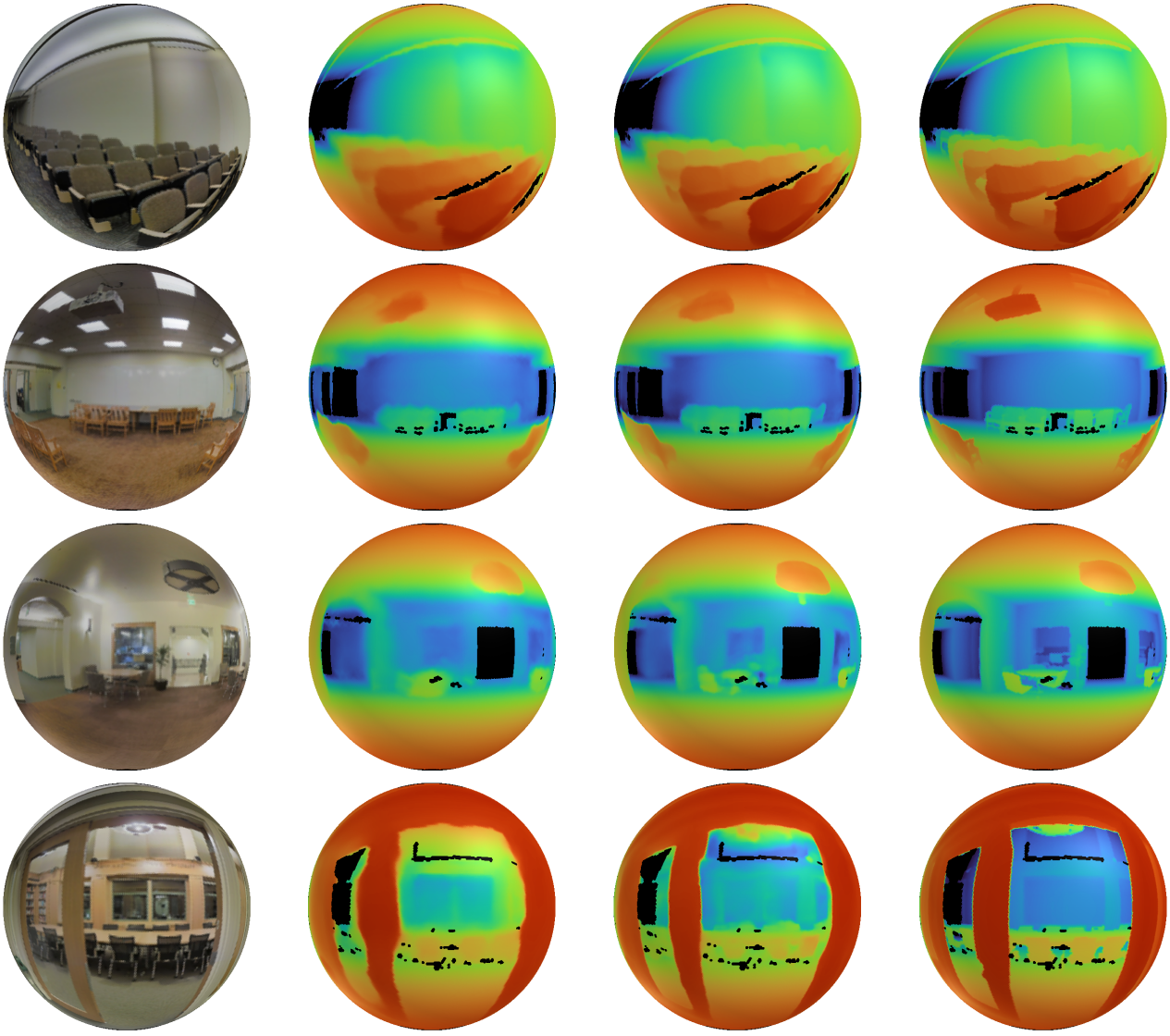

Figure 13. **High resolution results. From left to right:** RGB, Rank 7, Rank 8, GT. Results in rank 8 are sharper and visibly more accurate in their depth estimation.

C.3. Qualitative High-Resolution Comparison

In Fig. 13, we show the depth estimation results of rank 7 and rank 8 models. As specified in Appendix B.1, the rank 8 model was finetuned on rank 7 pretrained weights. Beyond the better quantitative performence shown in Tab. 5 (in the main paper), these results show that the higher resolution model produces a sharper and more accurate depth image.

References

[1] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105*, 2017. 2, 1, 5

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[3] Jia Zheng, Junfei Zhang, Jing Li, Rui Tang, Shenghua Gao, and Zihan Zhou. Structured3d: A large photo-realistic dataset for structured 3d modeling. In *Proceedings of The European Conference on Computer Vision (ECCV)*, 2020. 2, 1, 5