Adaptive Spot-Guided Transformer for Consistent Local Feature Matching ——Supplementary Material——

Jiahuan Yu[1*], Jiahao Chang[1*], Jianfeng He[1], Tianzhu Zhang[1,2†], Jiyang Yu[3], Feng Wu[1]
[1]University of Science and Technology of China, [2]Deep Space Exploration Lab, [3]China Academy of Space Technology

In this supplementary material, we first introduce the general sparse attention operator in Section 1. In Section 2, we provide some details about our experiment. In Section 3, we show additional visualizations about the spot-guided attention and adaptive scaling modules.

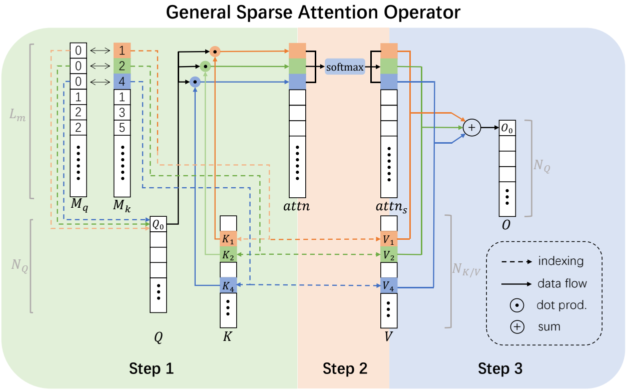

Figure 1. The illustration of our general sparse attention operator.

1. General Sparse Attention Operator

Due to irregular key/value token number for each query in Spot Attention, the naive implementation by PyTorch [7] is not efficient for memory and computation, which uses a mask to set unwanted values in the attention map to 0. More generally, the same problem also exists when the numbers of key corresponding to queries are not the same. Inspired by PointNet [8] and Stratified Transformer [4], we implement a general sparse attention operator using CUDA that is efficient in terms of memory and computation. We attempt to only compute the necessary attention between much less query/key tokens.

We can divide a vanilla attention operator into 3 steps. Inputs are grouped as query Q, key K and value V. First, the attention map A is computed by dot production as $A = QK^T$. Then, a softmax operator is performed on the attention map: $A_s = \text{softmax}(A/\sqrt{d_k})$. Finally, the updated query O can be obtained by $O = A_sV$. We optimize these three steps separately.

In the step 1, because only a few results in A are useful for sparse attention, we do not need to compute the full A. Instead, we compute the dot productions between L_m pairs of query and key. M_q and M_k record the indexes of query and key tokens whose dot productions are needed. The length of M_q and M_k are both L_m. Here, we denote the sparse attention map as $attn$, which is calculated by

$$attn[i] = Q[M_q[i]]K[M_k[i]]^T, \ i = 0, 1, \cdots, L_m - 1. \quad (1)$$

In the step 2, we group the elements in $attn$ with the same query index and apply softmax on each group. The result is denoted as $attn_s$.

In the step 3, we compute the updated query

$$O[q] = \sum_{M_q[i]=q} attn_s[i] \cdot V[M_k[i]]. \quad (2)$$

All of three steps are implemented in CUDA.

Compared with the naive implementation using PyTorch [7], our highly optimized implementation reduces the memory and time complexity from $\mathcal{O}(N_q \cdot N_k \cdot N_h \cdot N_d^2)$ to $\mathcal{O}(L_m \cdot N_h \cdot N_d^2)$, where N_q, N_k and N_h are separately the numbers of query tokens, key tokens and attention heads, and N_d is the dimension of each head. Considering $L_m \ll N_q \cdot N_k$, our implementation is much more efficient than the naive implementation.

In particular, we also calculate the matching matrix in spot-guided attention in this way and set the probability of unrelated pixels to 0, which can greatly reduce the memory and computation cost.

2. Experimental Details

2.1. Training Details

To reduce the GPU memory, we randomly sample 50% of ground truth matches to supervise the matching matrix at the coarse stage. And we sample 20% of the maximum number of coarse-level possible matches at the fine stage.

[*] Equal Contribution
[†] Corresponding Author

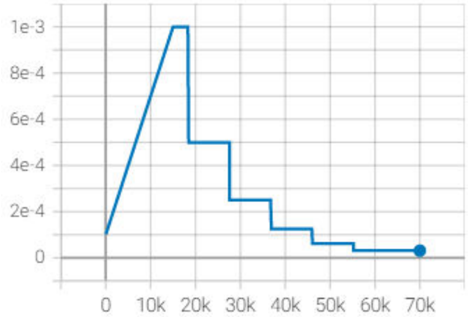

Figure 2. Learning rate curve while training on MegaDepth [5].

We train ASTR on MegaDepth [5] for 15 epochs. The initial learning rate is 1×10^{-3}, with a linear learning rate warm-up for 15000 iterations. The learning rate curve is shown in Figure 2.

2.2. Differences between Baseline and LoFTR

There are two main differences between our baseline and LoFTR [9].

(1) Normalized Positional Encoding. LoFTR [9] adopts the absolute sinusoidal positional encoding by following [1]:

$$\mathrm{PE}_i(x,y) = \begin{cases} \sin(w_k \cdot x), & i = 4k \\ \cos(w_k \cdot x), & i = 4k+1 \\ \sin(w_k \cdot y), & i = 4k+2 \\ \cos(w_k \cdot y), & i = 4k+3 \end{cases}, \tag{3}$$

where $w_k = \frac{1}{10000^{2k/d}}$, d denotes the number of feature channels and i is the index for feature channels. Considering the gap in image resolution between training and testing, we utilize the normalized positional encoding as [2], which is proven to mitigate the impact of image resolution changes in [2]. The normalized positional encoding $\mathrm{NPE}_i(\cdot,\cdot)$ can be expressed as

$$\mathrm{NPE}_i(x,y) = \mathrm{PE}_i(x * \frac{W_{train}}{W_{test}}, y * \frac{H_{train}}{H_{test}}), \tag{4}$$

where $W_{train/test}$ and $H_{train/test}$ are width and height of training/testing images.

(2) Convolution in Attention. Chen et al. [2] find that replacing the self attention with convolution can improve the performance. Hence, we deprecate self attention and MLP, and utilize a 3 × 3 convolution in our ASTR.

2.3. CNN Backbone

Here we leverage a deepened version of Feature Pyramid Network (FPN) [6], which achieves a minimum resolution of 1/32. The initial dimension for the stem is still 128 as LoFTR [9], and the number of feature channels for subsequent stages is [128, 196, 256, 256, 256].

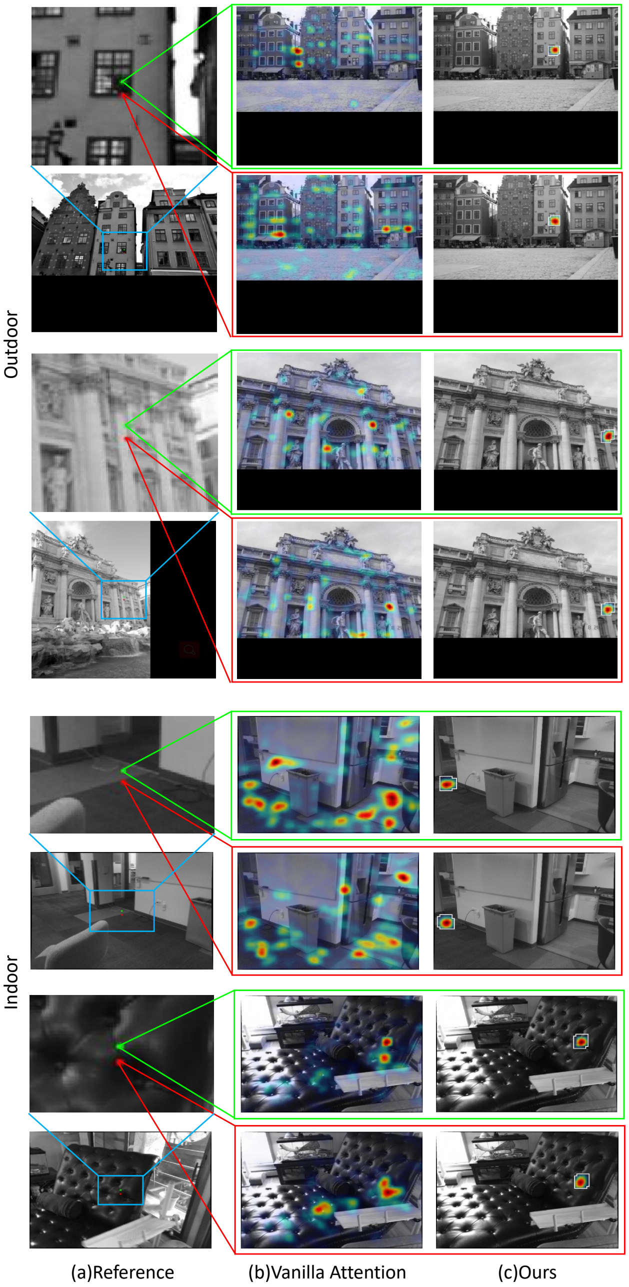

Figure 3. Visualizations of vanilla and spot-guided attention maps on MegaDepth [5] (outdoor) and ScanNet [3] (indoor).

3. Visualization Results

In Figure 3, we pick up two similar adjacent pixels as queries and visualize the corresponding attention maps of vanilla and our spot-guided attention for comparison. The vanilla attention mechanism is vulnerable to repetitive textures, while our spot-guided attention can focus on the correct areas in these repeated texture regions. Because large scale variation occurs frequently on outdoor datasets, we

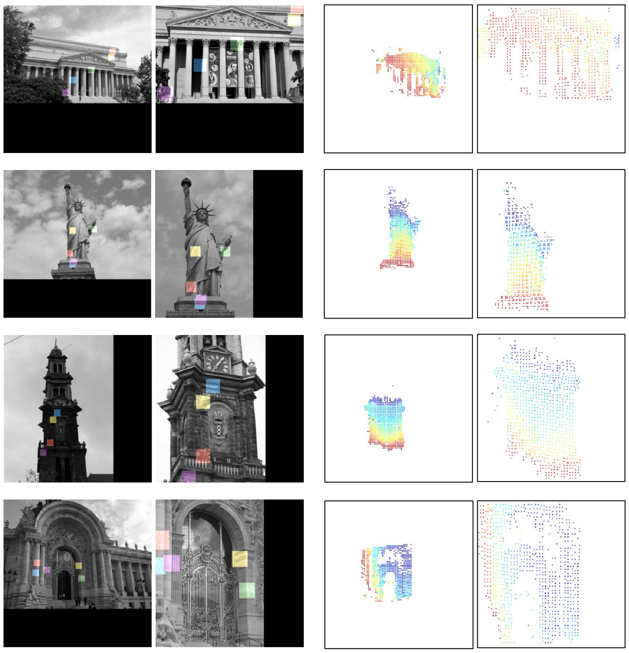

Figure 4. Visualizations of grids from adaptive scaling module and corresponding depth maps on MegaDepth [5]. Note that we use depth values with scale uncertainty to compose the depth maps.

mainly visualize the grids from the adaptive scaling module and corresponding depth maps on MegaDepth [5]. As shown in Figure 4, our adaptive scaling module can adjust the size of grids according to depth information.

References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 2

[2] Hongkai Chen, Zixin Luo, Lei Zhou, Yurun Tian, Mingmin Zhen, Tian Fang, David Mckinnon, Yanghai Tsin, and Long Quan. Aspanformer: Detector-free image matching with adaptive span transformer. *arXiv preprint arXiv:2208.14201*, 2022. 2

[3] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017. 2

[4] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified transformer for 3d point cloud segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8500–8509, 2022. 1

[5] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2041–2050, 2018. 2, 3

[6] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017. 2

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019. 1

[8] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 1

[9] Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. Loftr: Detector-free local feature matching with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8922–8931, 2021. 2