Supplementary Materials

Atsuya Nakata and Takao Yamanaka

Sophia University, Tokyo, Japan
a-nakata-7r0@eagle.sophia.ac.jp, takao-y@sophia.ac.jp

A Source codes of network architecture

The source codes for the network architecture in the proposed method are shown in Program 1.1 and Program 1.2. Program 1.1 is the source code of the low-resolution model at the first stage, where MultiAxisTransformerLayer is the MaxViT layer provided in Program 1.2. The input x is added with the positional encoding and the feature map of the conditional image compressed by CNN. The source code of the high-resolution model is almost same as Program 1.1, except that the feature map of the NFoV image extracted from the low-resolution image at the first stage is additionally added to the input x.

B Comparison of models at first and second stages

The proposed method adopted MaxViT models both at the first and second stages for the low-resolution and high-resolution models. The different models were also examined for the models, as shown in Table 4. (1) is the proposed method using MaxViT at both stages. In (2), MaxViT at the first stage was replaced with Transformer. In (3), MaxViT at the second stage was replaced with MultiAxisTransformer [21]. In (4), Grid Attention in MaxViT at the second stage was replaced with the neighbor attention in Fig. 10. Although IS and LPIPS were highest with (4), FID was greatly improved with (1) where MaxViT was used at both stages. Therefore, the model (1) was adopted as the proposed method in this paper.

C Sample images of proposed method compared with conventional method

Additional sample images of the synthesized omni-directional images in Fig. 7 are shown in Fig. 11. A comparison of synthesized omni-directional images between 2S-ODIS and 2S-ODIS 2day are shown in Fig. 12. Furthermore, additional sample images of the NFoV images toward the ground in Fig. 8 are shown in Fig. 13.

Program 1.1: Source code of MaxViT at first stage in PyTorch

```
class MultiAxisTransformer(nn.Module):
    def __init__(self, vocab_size, d_model,layer=8,seq_len=512):
        # vocab_size: The number of VQGAN Codebooks
        # d_model: The number of dimentions of model
        # layer: The number of MaxViTLayers
        # seq_len: The resolution of the input latent variables
        super(MultiAxisTransformerModel, self).__init__()
        self.positional_encoding = nn.Parameter(torch.randn(1,seq_len,d_model))
        self.embedding = nn.Embedding(vocab_size+1, d_model)
        self.trans_layers = nn.ModuleList([MultiAxisTransformerLayer(d_model,8) for i in range(layer)])
        self.mask_condition_conv = CNN(d_model)
        self.fc = nn.Linear(d_model, vocab_size)

    def forward(self, x, masked_condition,*args):
        x = self.embedding(x)
        condition = einops.rearrange(self.mask_condition_conv(masked_condition),"b c h w->b (h w) c")
        x = x + self.positional_encoding + condition
        for layer in self.trans_layers:
            x = layer(x)
        output = self.fc(x)
        return output
```

Program 1.2: Source code of MaxViT layer in PyTorch

```
class MultiAxisTransformerLayer(nn.Module):
    def __init__(self, d_model, n_head, patch_size=4):
        # d_model: The number of dimentions of model
        # n_head: Number of attention heads
        # patch_size: The number of patch size of block attention

        super().__init__()
        self.mbconv = MBConvBlock(d_model)
        self.block_attention = nn.TransformerEncoderLayer(d_model, n_head,dim_feedforward=d_model*4, batch_first=True,norm_first=True)
        self.grid_attention = nn.TransformerEncoderLayer(d_model, n_head,dim_feedforward=d_model*4, batch_first=True,norm_first=True)
        self.patch_size = patch_size

    def forward(self,x):
        x = einops.rearrange(x, "b (h w) c->b c h w",h=16)
        b,c,h,w = x.shape
        x = self.mbconv(x)
        x = einops.rearrange(x,"b c (h1 h2) (w1 w2)->(b h1 w1) (h2 w2) c",
            h1=h//self.patch_size,h2=self.patch_size,
            w1=w//self.patch_size,w2=self.patch_size)
        x = self.block_attention(x)
        x = einops.rearrange(x,"(b h1 w1) (h2 w2) c->(b h2 w2) (h1 w1) c",
            h1=h//self.patch_size,h2=self.patch_size,
            w1=w//self.patch_size,w2=self.patch_size)
        x = self.grid_attention(x)
        x = einops.rearrange(x,"(b h2 w2) (h1 w1) c->b (h1 h2 w1 w2) c",
            h1=h//self.patch_size,h2=self.patch_size,
            w1=w//self.patch_size,w2=self.patch_size)
        return x
```

Table 4: Comparison of models at first and second stages

First Stage	Second Stage	IS (↑)	FID (↓)	LPIPS (↑)
(1) MaxViT	MaxViT	5.969	**18.263**	0.662
(2) Transformer	MaxViT	6.010	23.168	0.660
(3) MaxViT	MultiAxisTransformer [21]	5.820	22.274	0.657
(4) MaxViT	NeighborAttention (Fig. 10)	**6.030**	20.529	**0.667**

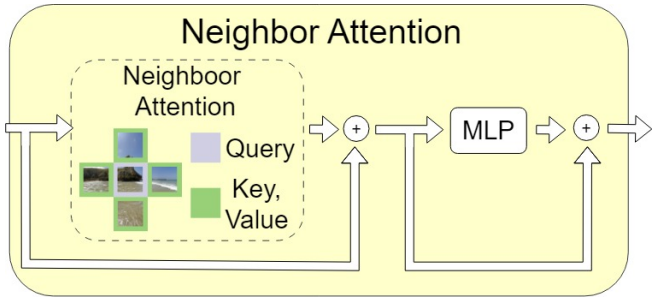

Fig. 10: Neighbor attention. The attention is calculated with adjacent NFoV images

References

21. Zhao, L., Zhang, Z., Chen, T., Metaxas, D., Zhang, H.: Improved transformer for high-resolution gans. In: Advances in Neural Information Processing Systems(NeurIPS). vol. 34, pp. 18367–18380 (2021)

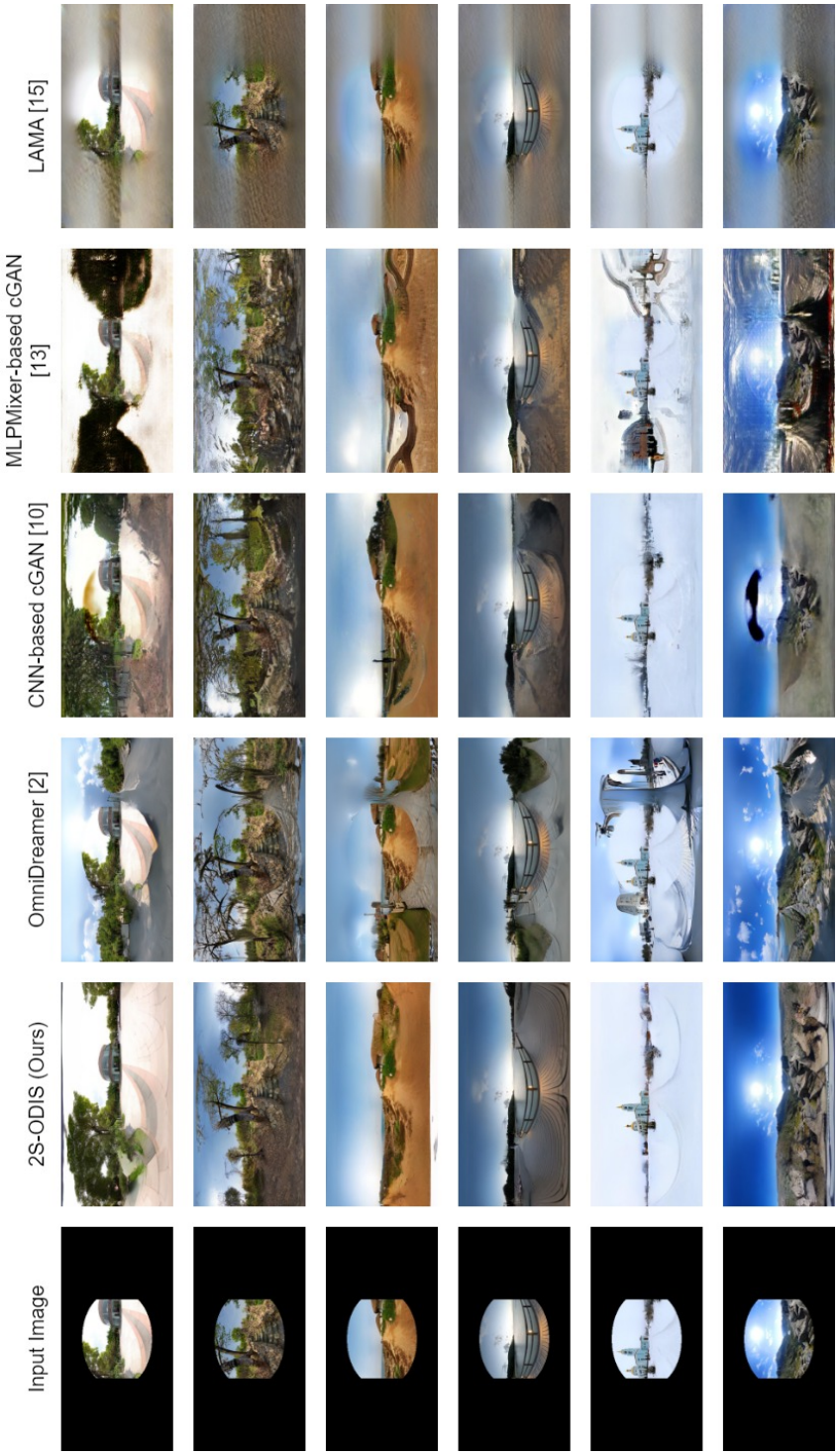

Fig. 11: Additional examples of synthesized omni-directional images in Fig. 7

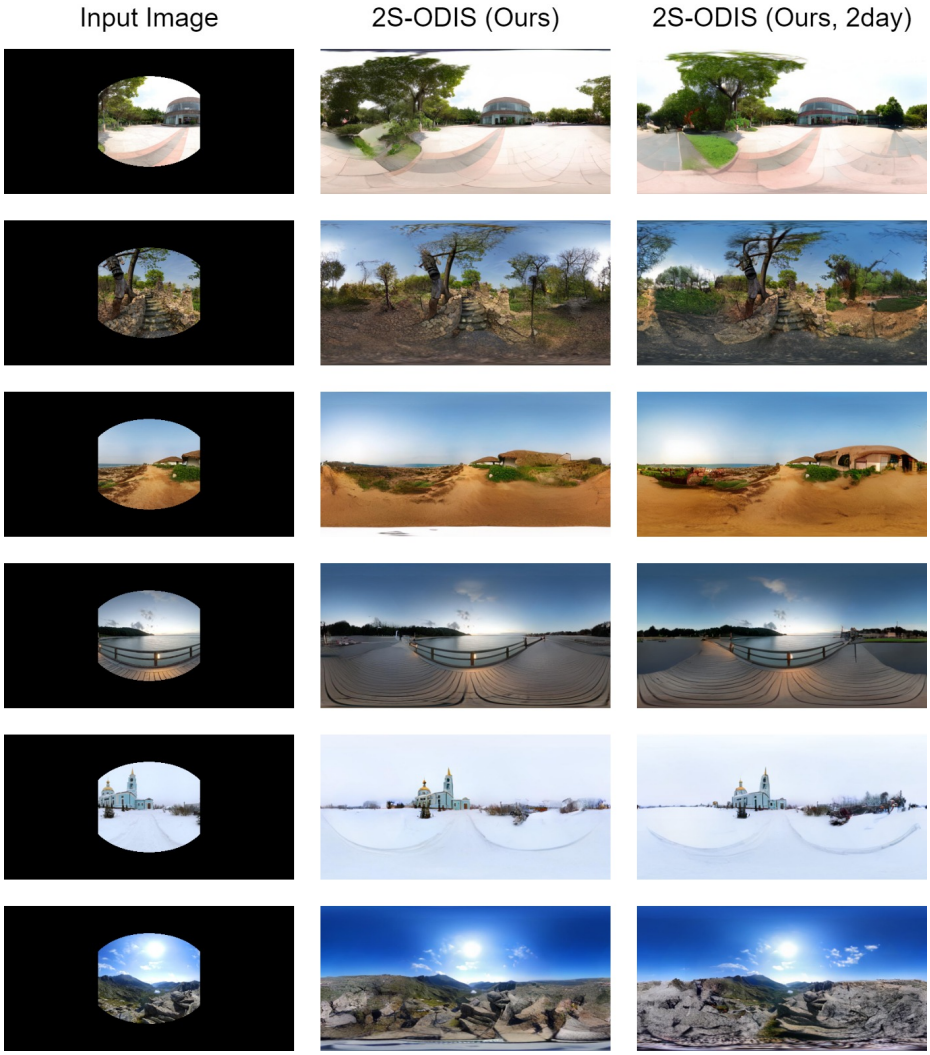

Fig. 12: A comparison of synthesized omni-directional images between 2S-ODIS and 2S-ODIS 2day

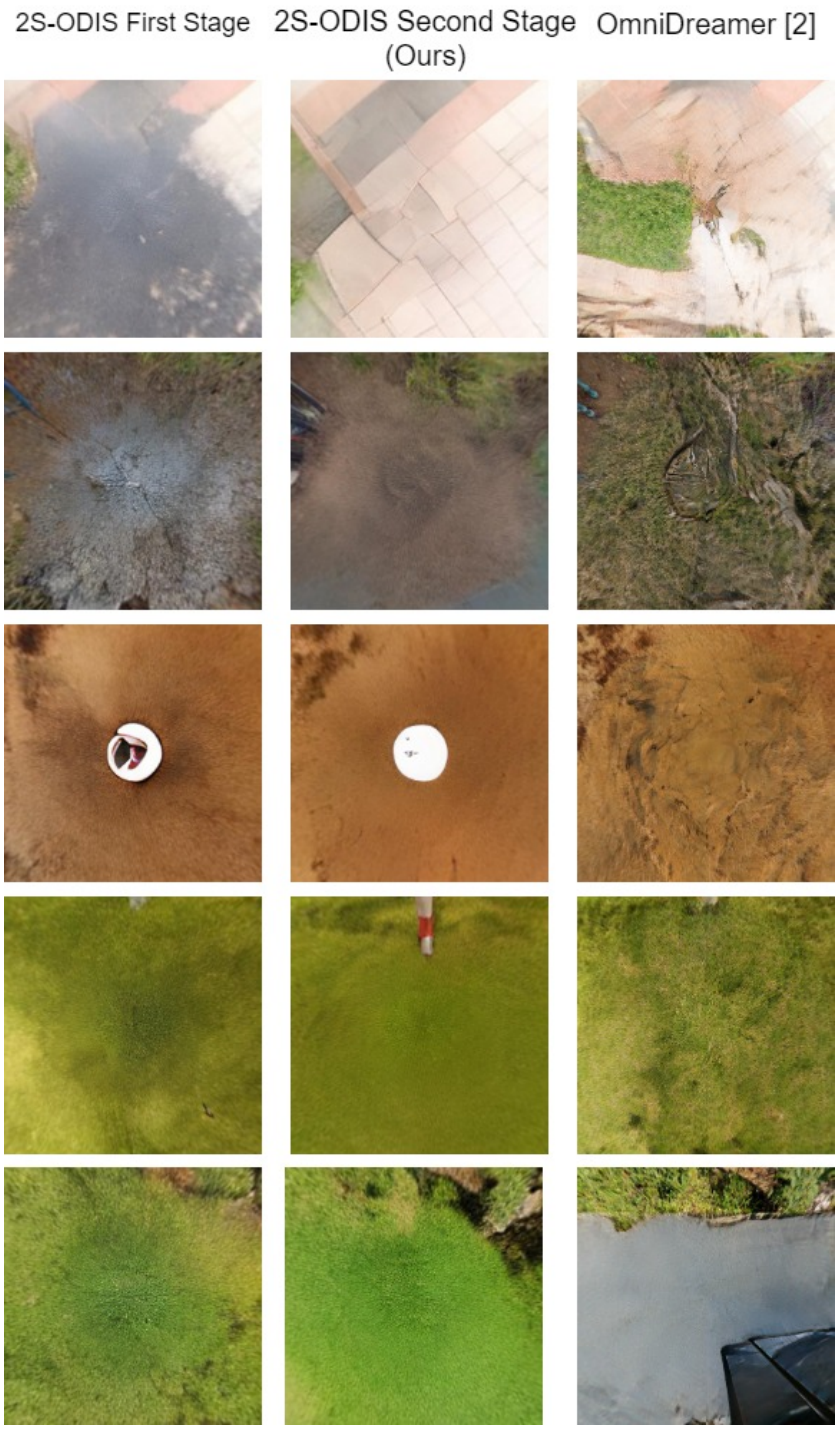

Fig. 13: Additional examples of NFoV images toward ground extracted from synthesized omni-directional images in Fig. 8.