Graph Representation for Order-aware Visual Transformation (Supplementary Material)

In this supplementary material, we provide additional implementation details, including details of the proposed model VTGen, the loss function, and training parameters used in the experiments to supplement our main submission. We also provide additional information on the dataset setup and examples for all change patterns used in the proposed OVT dataset.

A. Implementation Details

Transformation Encoder. Given the input of two images, the transformation encoder extracts the correlations between different image regions between the two images. We adopted a standard transformer encoder structure in the implementation of the transformer self-attention operation $\mathrm{Attention}\,(\mathrm{I}', \mathrm{I}', \mathrm{I}')$ described in subsection 4.2 of the main submission. More specifically, we implemented the transformer encoder with two layers, four heads, and a feedforward dimension of 2048.

Cross-attention Decoder. This decoder aims to associate image features with change queries to identify individual changes. Each change is described in ten words, which indicate the change type, the attributes of the changed object (color, material, shape), the before-change position (described by color, material, shape, or "ground ground ground", or "none none none" for add change), and the after-change position (with three words similar to those used in the before-change position, where "none none none" is for delete change), respectively. We also added a change consisting of the word "start" (ten times) to indicate the start of the change query. Since the max change number of the proposed OVT dataset is four, we set the change query number to five. For scenes with less than four changes, we added changes consisting of the word "none" (ten times) to reach the change query length.

For each image pair consisting of multiple changes, we first embedded each change to a $1 \times D$-dimensional feature q. More specifically, we used the Pytorch [1] nn.Embedding function and a standard transformer multi-head attention layer to encode the ten words to a $10 \times D$-dimensional feature and then summed up the features of all ten words, which results in a $1 \times D$-dimensional change query. During the training process, we used the change queries obtained from ground truth change information. During the testing process, we first input the "start" change and then updated the change query by the model output. For the cross-attention operation $\mathrm{Attention}\,(\mathrm{Q}, \mathrm{I}'', \mathrm{I}'')$ described in subsection 4.2 in the main submission, we used a standard transformer with two layers, four heads, and a 2048 feedforward dimension.

Change Content Decoder. This encoder aims to find the correlations between different changes and determine the detailed change contents for each change. We experimented with two structures, including a transformer and a GCN for the change content decoder. We implemented a standard transformer with a feedforward dimension of 2048 and experimented with varied heads and layers. We implemented GCN with MLP layers. For each layer of the GCN, we first concatenated each pair of input feature $\{q'_1, q'_2, ..., q'_n\}$ and added a two-layer MLP to encode the paired features, resulting $\{p_{1,2}, p_{1,3}, ..., p_{n-1,n}\}$, where $p_{j,k} \in \mathbb{R}^D$. Next, we updated features by $q''_i = \sum_{j=i \,\vee\, k=i} p_{j,k}$ and then summed up the obtained features with the original input features of the current GCN layer to prevent over-fitting. During the ablation study, we implemented the GCN with varied layers. Finally, we used ten classification heads to identify the detailed change contents of each change, where each classification head consists of two fully-connection layers along with a softmax function.

Change Oder Decoder. We implemented the change order decoder with the same structure as the change content decoder, except for the input feature dimension and the classification heads. We added a classification head to identify the presence of the directed edges between each pair of two changes.

Loss Function. We compute the loss for nodes (change contents) and edges separately, and equation (1) in the main paper shows the change content part. In the case of a graph-matching loss setup, we generate changes in a specific order, and then compute the minimal loss across all possible orders of changes. On the other hand, for the cross-entropy loss setup, we solely calculate and assess the loss using the predetermined order of changes provided in the input, disregarding other potential change orders resulting from synchronous changes.

Training Parameters. During the experiments on the proposed OVT dataset, we jointly trained all models together, including the transformation encoder, cross-attention decoder, change content decoder, and change order decoder. During the experiments on the existing dataset CLEVR-Multi-Change [2], we removed the change order decoder and jointly trained the remaining three models. During all experiments, we set the batch size to 64 and used the Adam optimizer [3].

B. Dataset Setup

The OVT dataset consists of three basic types of changes: “add”, “delete”, and “move”. The “add” and “delete” changes involve adding a new object or removing an existing one from a visible location within the scene, respectively. The “move” change is created by relocating an object from its initial coordinates to a new location. The minimal distance between two objects (excluding stacked objects) was set to 1.45, while the side length and diameter for cubes, spheres, and cylinders were set to 0.8.

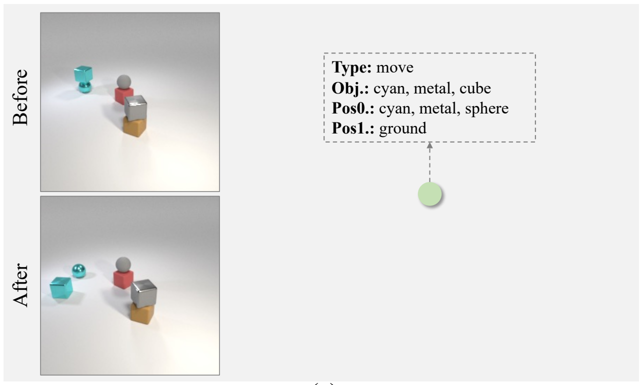
This paper discusses the recognition of order-aware changes from image pairs consisting of asynchronous and synchronous changes between two images. In the proposed OVT dataset, we set the maximum number of changes within two images to four. We included a total of 15 change patterns with specific change numbers and orders in the OVT dataset. Examples of each change pattern are shown in Figure 6 and Figure 7.

References

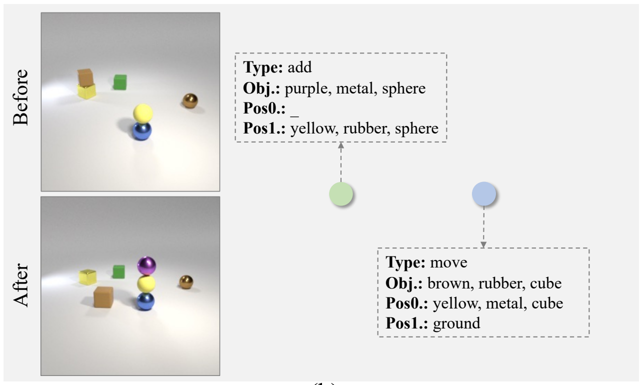
[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 1

[2] Yue Qiu, Shintaro Yamamoto, Kodai Nakashima, Ryota Suzuki, Kenji Iwata, Hirokatsu Kataoka, and Yutaka Satoh. Describing and localizing multiple changes with transformers. In *ICCV*, 2021. 2

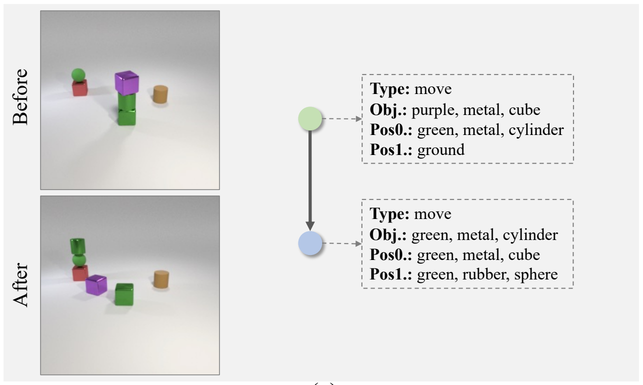
[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 2



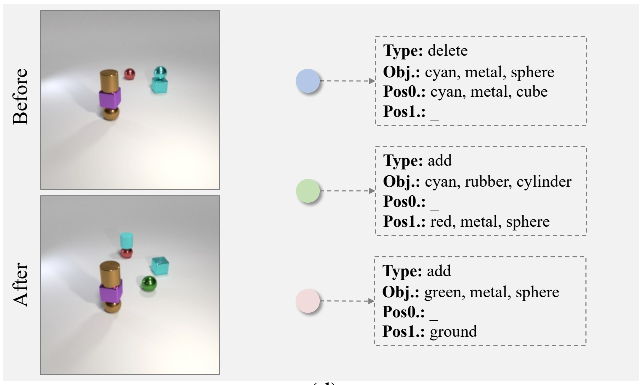
(a)


Before
After
Type: add
Obj.: purple, metal, sphere
Pos0.: _
Pos1.: yellow, rubber, sphere
Type: move
Obj.: brown, rubber, cube
Pos0.: yellow, metal, cube
Pos1.: ground

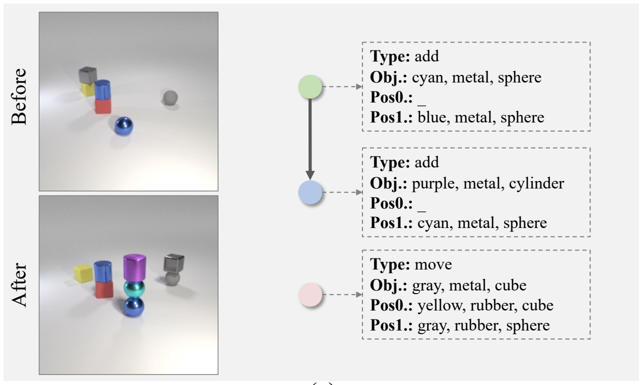
(b)


Before
After
Type: move
Obj.: purple, metal, cube
Pos0.: green, metal, cylinder
Pos1.: ground
Type: move
Obj.: green, metal, cylinder
Pos0.: green, metal, cube
Pos1.: green, rubber, sphere

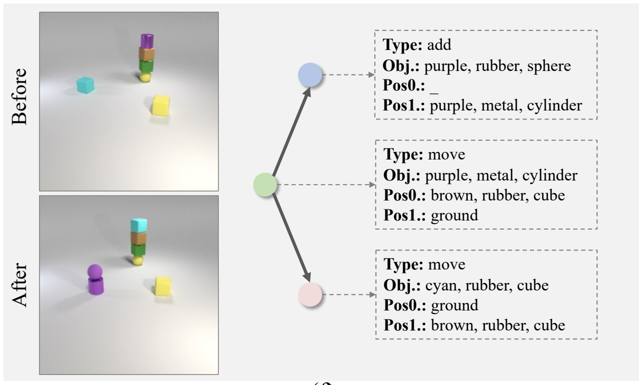
(c)


Before
After
Type: delete
Obj.: cyan, metal, sphere
Pos0.: cyan, metal, cube
Pos1.: _
Type: add
Obj.: cyan, rubber, cylinder
Pos0.: _
Pos1.: red, metal, sphere
Type: add
Obj.: green, metal, sphere
Pos0.: _
Pos1.: ground

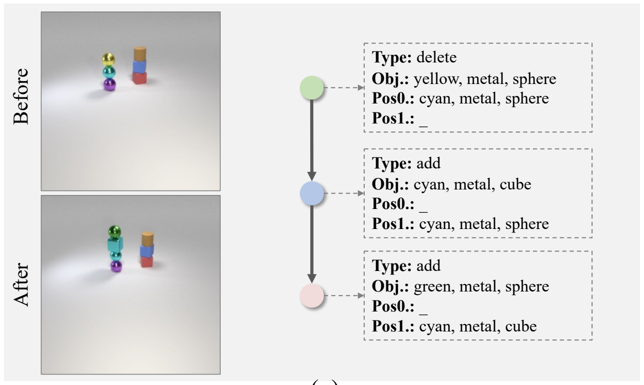
(d)


Before
After
Type: add
Obj.: cyan, metal, sphere
Pos0.: _
Pos1.: blue, metal, sphere
Type: add
Obj.: purple, metal, cylinder
Pos0.: _
Pos1.: cyan, metal, sphere
Type: move
Obj.: gray, metal, cube
Pos0.: yellow, rubber, cube
Pos1.: gray, rubber, sphere

(e)

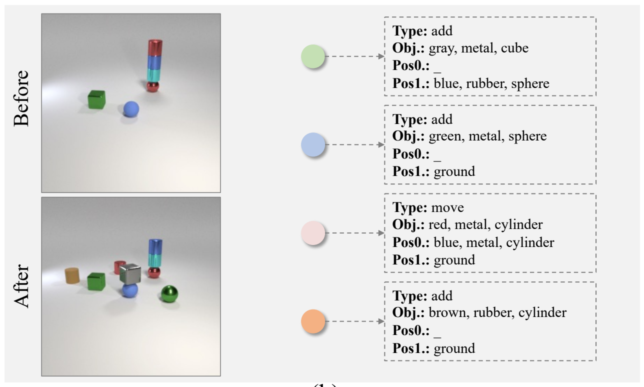

Before
After
Type: add
Obj.: purple, rubber, sphere
Pos0.: _
Pos1.: purple, metal, cylinder
Type: move
Obj.: purple, metal, cylinder
Pos0.: brown, rubber, cube
Pos1.: ground
Type: move
Obj.: cyan, rubber, cube
Pos0.: ground
Pos1.: brown, rubber, cube

(f)


Before
After
Type: delete
Obj.: yellow, metal, sphere
Pos0.: cyan, metal, sphere
Pos1.: _
Type: add
Obj.: cyan, metal, cube
Pos0.: _
Pos1.: cyan, metal, sphere
Type: add
Obj.: green, metal, sphere
Pos0.: _
Pos1.: cyan, metal, cube

(g)

Figure 6. All change patterns (a-g) for scenes with one to three changes used in the OVT dataset. Here, the change pattern indicates the specific number and order of changes. We use circular shapes to indicate changes, with details recorded in dashed boxes and solid gray arrows to indicate the chronological order between changes.



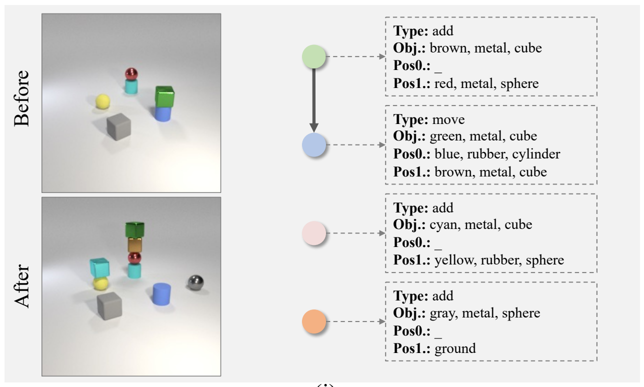
Type: add
Obj.: gray, metal, cube
Pos0.: _
Pos1.: blue, rubber, sphere

Type: add
Obj.: green, metal, sphere
Pos0.: _
Pos1.: ground

Type: move
Obj.: red, metal, cylinder
Pos0.: blue, metal, cylinder
Pos1.: ground

Type: add
Obj.: brown, rubber, cylinder
Pos0.: _
Pos1.: ground

(h)



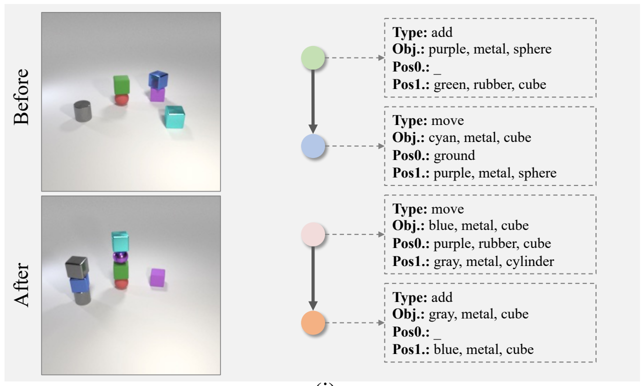
Type: add
Obj.: brown, metal, cube
Pos0.: _
Pos1.: red, metal, sphere

Type: move
Obj.: green, metal, cube
Pos0.: blue, rubber, cylinder
Pos1.: brown, metal, cube

Type: add
Obj.: cyan, metal, cube
Pos0.: _
Pos1.: yellow, rubber, sphere

Type: add
Obj.: gray, metal, sphere
Pos0.: _
Pos1.: ground

(i)



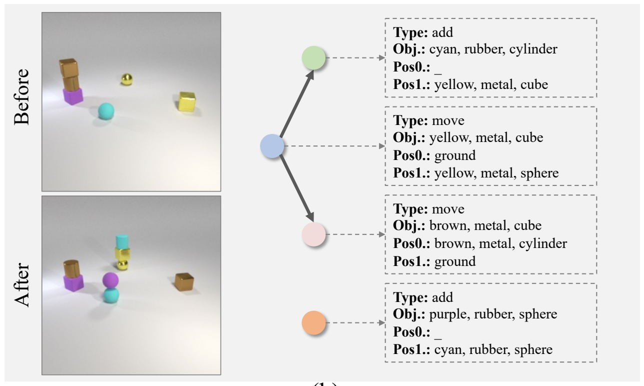
Type: add
Obj.: purple, metal, sphere
Pos0.: _
Pos1.: green, rubber, cube

Type: move
Obj.: cyan, metal, cube
Pos0.: ground
Pos1.: purple, metal, sphere

Type: move
Obj.: blue, metal, cube
Pos0.: purple, rubber, cube
Pos1.: gray, metal, cylinder

Type: add
Obj.: gray, metal, cube
Pos0.: _
Pos1.: blue, metal, cube

(j)



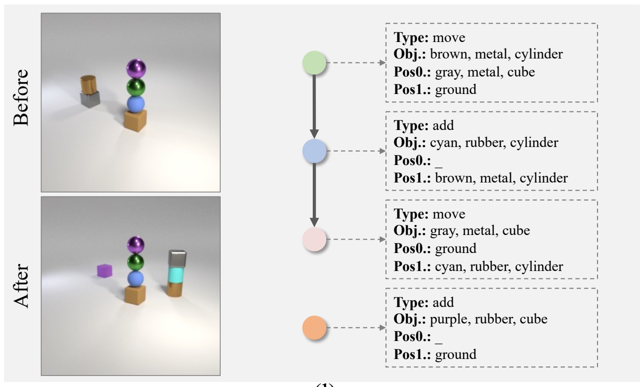
Type: add
Obj.: cyan, rubber, cylinder
Pos0.: _
Pos1.: yellow, metal, cube

Type: move
Obj.: yellow, metal, cube
Pos0.: ground
Pos1.: yellow, metal, sphere

Type: move
Obj.: brown, metal, cube
Pos0.: brown, metal, cylinder
Pos1.: ground

Type: add
Obj.: purple, rubber, sphere
Pos0.: _
Pos1.: cyan, rubber, sphere

(k)



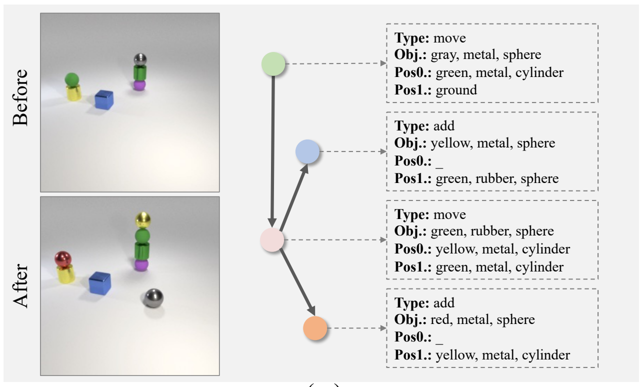
Type: move
Obj.: brown, metal, cylinder
Pos0.: gray, metal, cube
Pos1.: ground

Type: add
Obj.: cyan, rubber, cylinder
Pos0.: _
Pos1.: brown, metal, cylinder

Type: move
Obj.: gray, metal, cube
Pos0.: ground
Pos1.: cyan, rubber, cylinder

Type: add
Obj.: purple, rubber, cube
Pos0.: _
Pos1.: ground

(l)



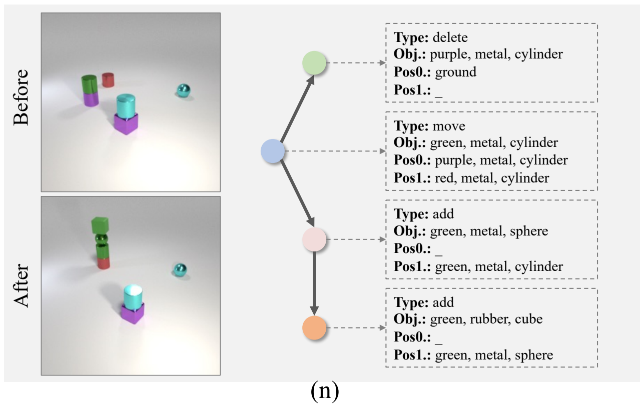
Type: move
Obj.: gray, metal, sphere
Pos0.: green, metal, cylinder
Pos1.: ground

Type: add
Obj.: yellow, metal, sphere
Pos0.: _
Pos1.: green, rubber, sphere

Type: move
Obj.: green, rubber, sphere
Pos0.: yellow, metal, cylinder
Pos1.: green, metal, cylinder

Type: add
Obj.: red, metal, sphere
Pos0.: _
Pos1.: yellow, metal, cylinder

(m)



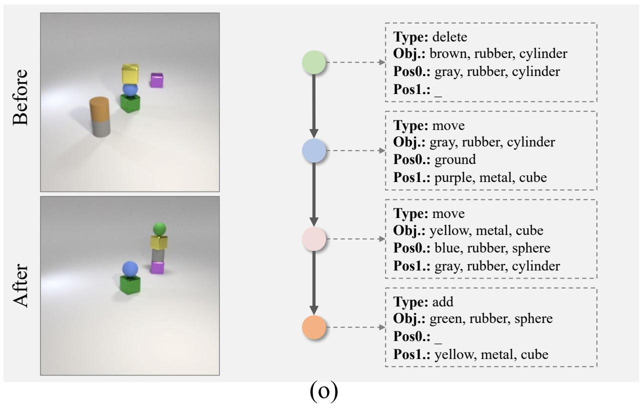
Type: delete
Obj.: purple, metal, cylinder
Pos0.: ground
Pos1.: _

Type: move
Obj.: green, metal, cylinder
Pos0.: purple, metal, cylinder
Pos1.: red, metal, cylinder

Type: add
Obj.: green, metal, sphere
Pos0.: _
Pos1.: green, metal, cylinder

Type: add
Obj.: green, rubber, cube
Pos0.: _
Pos1.: green, metal, sphere

(n)



Type: delete
Obj.: brown, rubber, cylinder
Pos0.: gray, rubber, cylinder
Pos1.: _

Type: move
Obj.: gray, rubber, cylinder
Pos0.: ground
Pos1.: purple, metal, cube

Type: move
Obj.: yellow, metal, cube
Pos0.: blue, rubber, sphere
Pos1.: gray, rubber, cylinder

Type: add
Obj.: green, rubber, sphere
Pos0.: _
Pos1.: yellow, metal, cube

(o)

Figure 7. All change patterns (h-o) for scenes with four changes used in the OVT dataset. Here, the change pattern indicates the specific number and order of changes. We use circular shapes to indicate changes, with details recorded in dashed boxes and solid gray arrows to indicate the chronological order between changes.