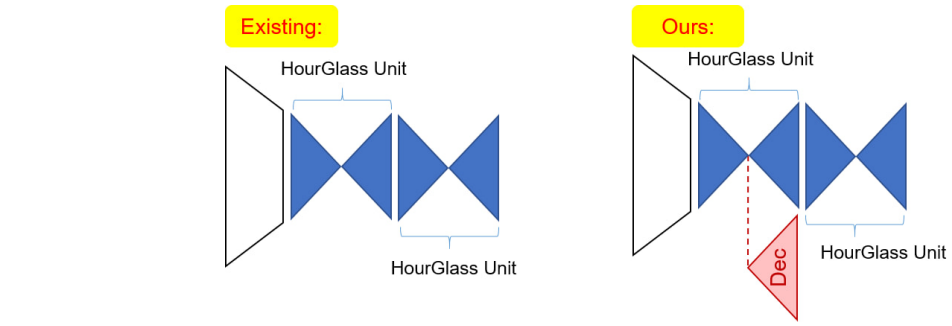
Supplementary Material: Hole-robust Wireframe Detection

Naejin Kong, Kiwoong Park, Harshith Goka
Samsung Research AI Center
{naejin.kong, kyoong.park, h9399.goka}@samsung.com

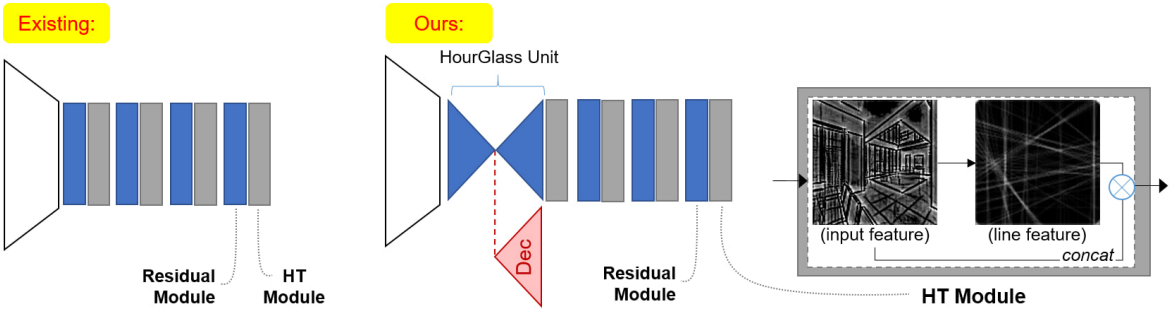
1. Introduction

In this supplementary material, we show more details on the approach and further analysis, experiments and results.

Contents



(a) **HG2** Architecture



(b) **HT-HG** Architecture

Figure 1: **Backbone architectures**. (a) **HG2** architecture. (Left): HourGlass units are stacked twice. (Right): Our RGB decoder is connected at the first bottleneck. (b) **HT-HG** architecture. (Left): A Hough-transform module is attached right after each Residual Module. See Figure 2 for details of the Residual Module. (Right): We replace the first Residual Module with a full HourGlass unit, and connect an RGB decoder at its bottleneck (see Figure 2 for details).

2. Details on RGB GAN

2.1. Backbone Network

The backbone network first downsamples a 512×512 input RGB image to 128×128 with 256 channels. Then, it stacks units adopted from HourGlass [9] as follows:

HG2 Architecture: A full HourGlass unit is adopted from [9]. In this backbone architecture, the HourGlass units are stacked up twice as shown in Figure 1(a): Existing. L-CNN [15], HAWP [12] and F-Clip(HG2) [1] frameworks use this **HG2** type backbone.

HT-HG Architecture: A Residual Module is a basic building block of the HourGlass network [9]. This module takes an input feature $\mathbf{x}$ in any channels and produces a residual output feature in $p * 2$ channels that is added to the input feature, where the original resolution of the input feature is preserved (see Figure 2 for details). In this backbone architecture, the Residual Modules are stacked four times, where a Hough-transfrom module (see Figure 1(b): Existing) is inserted right after each of the Residual Modules. HT-LCNN [15, 7] and HT-HAWP [12, 7] frameworks use this **HT-HG** type backbone.

2.2. RGB Decoder for RGB GAN

Our RGB decoder is connected at the bottleneck of the first HourGlass unit. Such a case on the **HG2** architecture is shown in Figure 1(a): Right. In case of the **HT-HG** architecture, we replace the first Residual Module with a full HourGlass unit and connect the RGB decoder at its bottleneck as shown in Figure 1(b): Right. We depict the RGB decoder architecture in detail in Figure 2. The last layer of the RGB decoder is a simple convolutional layer (one 1×1 convolution to compress channels and one batch normalization) that achieves a final 128×128 image in RGB channels. In practice, to better regularize the GAN training, we use spectral normalization [8] for the whole backbone network along with all branches, including the RGB decoder but excluding the Hough-transform modules, as well as a discriminator to be addressed in Section 2.3.

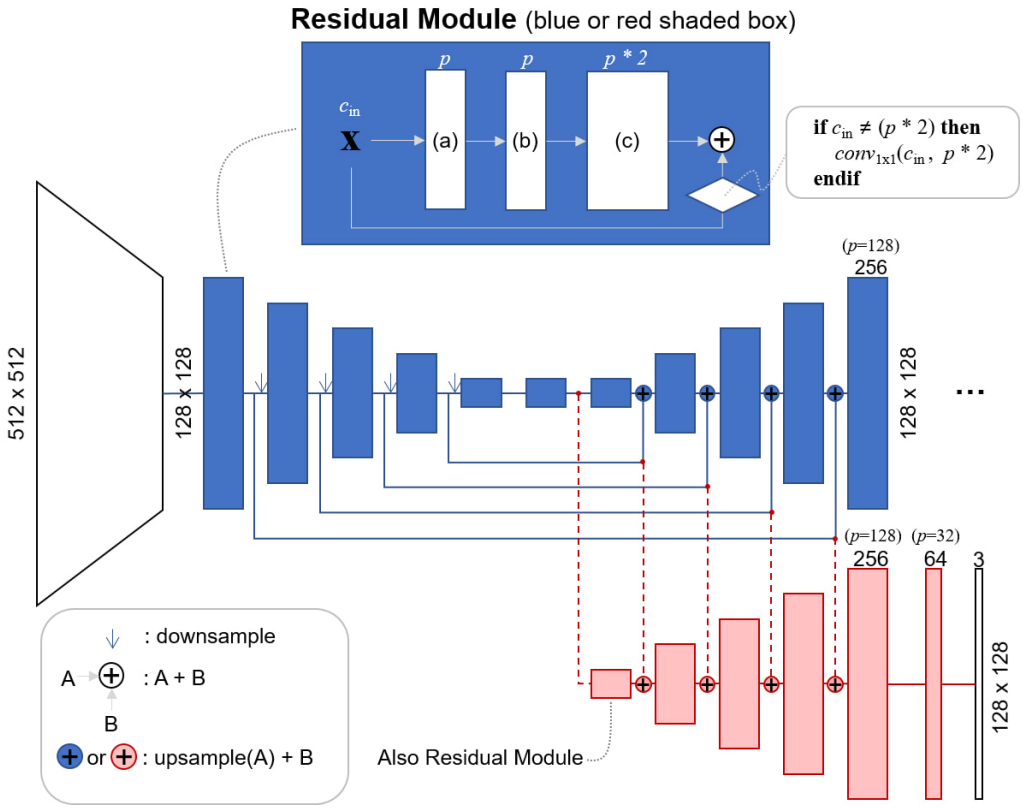


Figure 2: **First HourGlass unit (in blue) and RGB decoder (in red) in our RGB GAN.** The number above the box is the number of output channels. In the Residual Module shown on top, each of convolutional blocks (a), (b), (c) implies $batchnorm + ReLU + conv$, where (a) and (c) use $conv_{1x1}$ and (b) uses $conv_{3x3}$ with padding 1, hence the original resolution of the input feature is preserved. c_{in} is the number of channels of the input feature $\mathbf{x}$, and (a), (b), (c) blocks produce p, p, and $p * 2$ output channels, respectively. If $c_{in} \neq (p * 2)$, the number of channels of $\mathbf{x}$ is transformed to $p * 2$ before the residual addition. Each of the blue or red shaded boxes indicates a Residual Module shown on top of the figure. The final layer of the RGB decoder is a simple convolutional layer ($conv_{1x1}(64, 3) + batchnorm$) that outputs a 128×128 image in RGB channels.

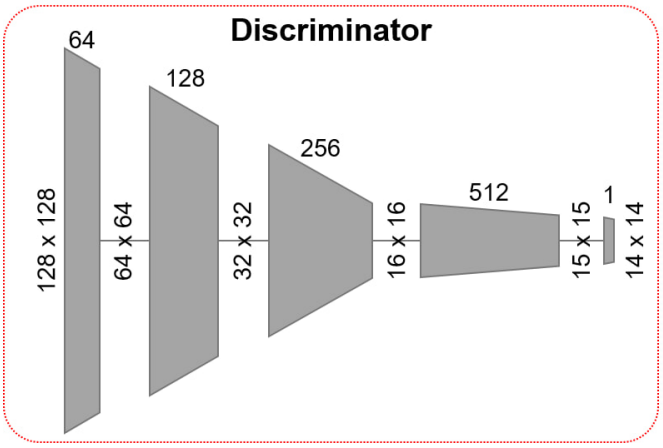


Figure 3: Discriminator architecture in our RGB GAN. The number above the trapezoid is the number of output channels.

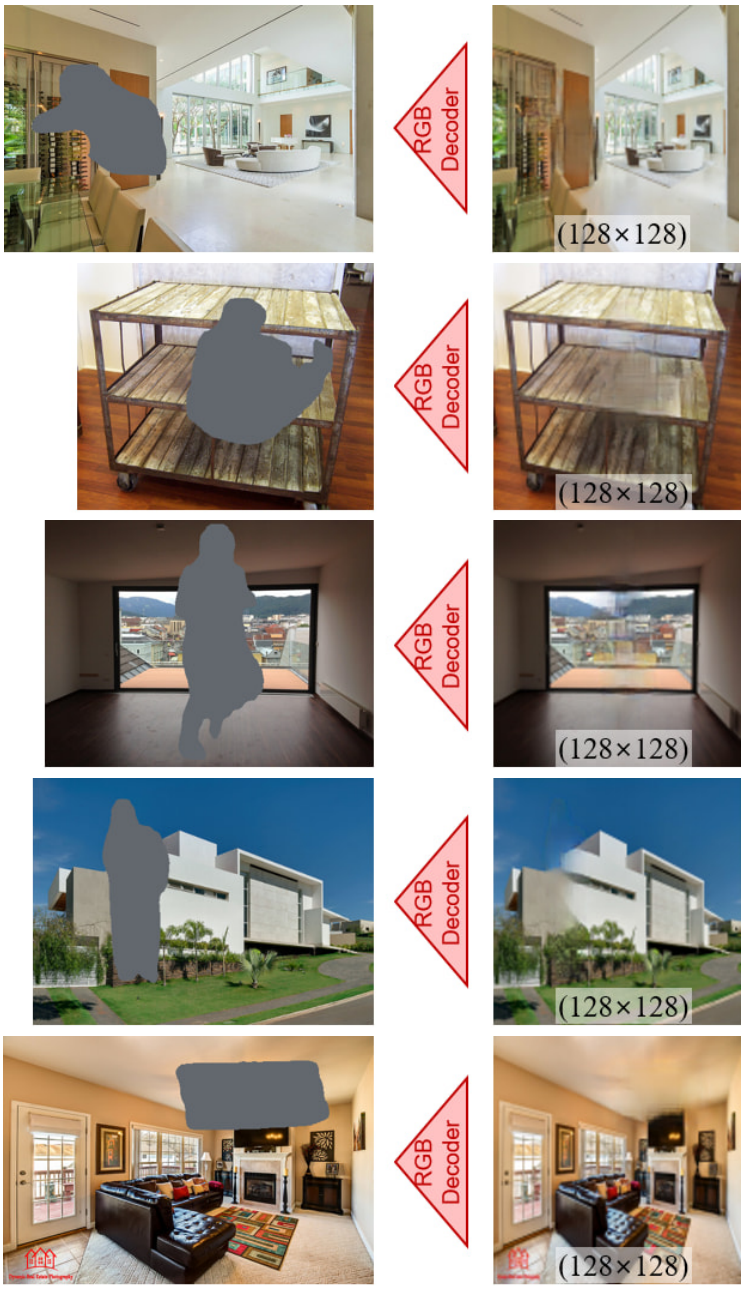


Figure 4: Generated RGB images through our RGB GAN.

2.3. Discriminator for RGB GAN

We depict the patch discriminator network architecture in Figure 3. We adapted it from the basic form in PatchGAN [6]. Each pixel in the final 14×14 output feature corresponds to a 70×70 patch in the input image. In practice, we use spectral normalization [8] for the discriminator, backbone and RGB decoder to better regularize the GAN training.

2.4. Generated RGB Images

We visualize in Figure 4 the generated RGB images in the 128×128 resolution from our trained RGB GAN. It demonstrates that the backbone network learned to infer the overall structure inside the hole, which will be then, propagated to more robust structural estimation through the other parts of the model.

3. Details on Conditional Training

3.1. Conditional Data Generation and Training

1. Creating a pool of object silhouettes

   (a) Preparing a pool of 1.3M object silhouette maps by applying Detectron2 [10] ('*things*' only from panoptic segmentation with `COCO-PanopticSegmentation/panoptic_fpn_R_50_3x`) on Places365 [14] Test set (328,500 images)

   (b) Leaving 1,107,482 object silhouette maps from the initial pool, where each silhouette map in the pool satisfies both a. and b. such that:
   a. `max(width, height)` of the tight bounding box around the silhouette map is smaller than 512 pixels.
   b. The silhouette map contains less than 78,643.2(= 512 * 512 * 30%) pixels.

   (c) According to the ratio of the hole size (i.e., number of pixels) to 512×512 pixels, grouping the object silhouette maps from the pool into one of 0.1-1%, 1-2%, . . . , 29-30% intervals, where each interval has 1% gap

   (d) Randomly selecting 1,500 object silhouette maps from each of the 0.1-1%, . . . , 29-30% intervals.

   (e) Finally, 45,000(= 1,500 silhouettes * 30 intervals) object silhouette maps are prepared.

2. Conditional data generation
   - Used both for **Avoid-isolation** and **Random Placement** training

   (a) Generating a pool of N *mask* maps for each input image in the training set (we use N = 10), and for each of the 0.1-1%, 1-2%, . . . , 9-10% hole size intervals (i.e., 10 intervals in total)

   (b) A *mask* map within a hole size interval is created as follows:
   a. Randomly select an object silhouette map within the interval size, from the silhouette pool made at Step 1.
   b. Superimpose the chosen silhouette onto a 512×512 region, by avoiding isolated components (see Algorithm 1 below).
   c. A 512×512 *mask* map with the silhouette hole is created.

   (c) Finally, for each image in the training set, N *mask* maps at each interval thus in total [N * 10 intervals] *mask* maps are prepared in advance.

3. Conditional training

   (a) Training by choosing a random size hole

      i. **Case1: Avoid-isolation.** For each image, randomly selecting one from all of the [N * 10 intervals] *mask* maps corresponding to the image
      Case2: Random Placement. For each image, randomly selecting one from all of the [N * 10 intervals * T images] *mask* maps, where T = number of all images in the dataset

      ii. Superimposing the hole in the *mask* map onto the 512×512 resized image to create a pair of ⟨*image_with_hole, mask*⟩

   (b) Training by progressively increasing the hole size along with epochs

      i. **Case1: Avoid-isolation.** For each image and at the designated hole size interval, randomly selecting one from the N *mask* maps corresponding to the image
      Case2: Random Placement. For each image and at the designated hole size interval, randomly selecting one from the [N * T images] *mask* maps, where T = number of all images in the dataset

      ii. Superimposing the hole in the *mask* map onto the 512×512 resized image to create a pair of ⟨*image_with_hole, mask*⟩

3.2. Avoid-Isolation Algorithm

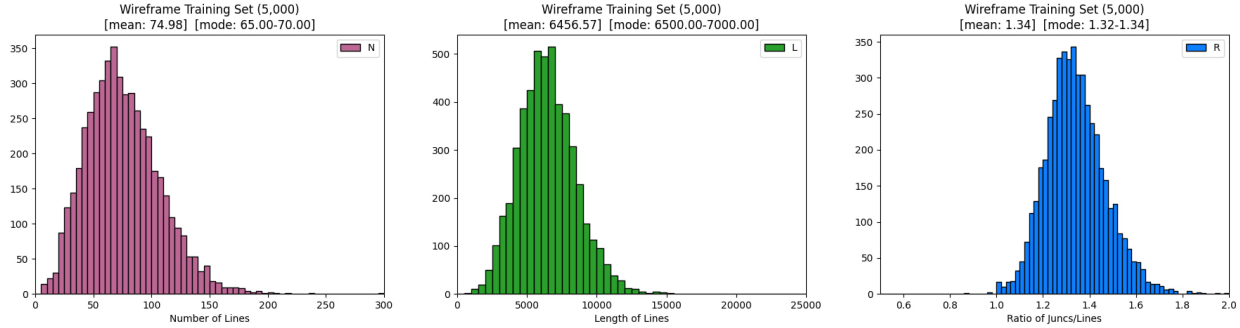
First, we define the following three hole types:

1. Hole Type 1:
    - The hole overlaps with line segment(s), and also does not contain any junction.
2. Hole Type 2:
    - One or more junction(s) are contained in the hole.
    - The number of line segments whose endpoints are both contained in the hole, is ≤ 1.
3. Hole Type 3:
    - One or more junction(s) are contained in the hole.
    - Satisfying the following criteria that can be made (sub-)optimal by repeated searching (see Algorithm 1: L10-20):
        - Fewest number of line segments whose endpoints are both contained in the hole
        - Largest total length of line segments that (at least partially) overlap with the hole

Note that Algorithm 1 attempts to create a mask by avoiding isolated components, but there can be exceptional cases that do not allow to strictly avoid isolation after all attempts, and in that case Hole Type 3 is used, which creates a mask that may contain some isolated components.

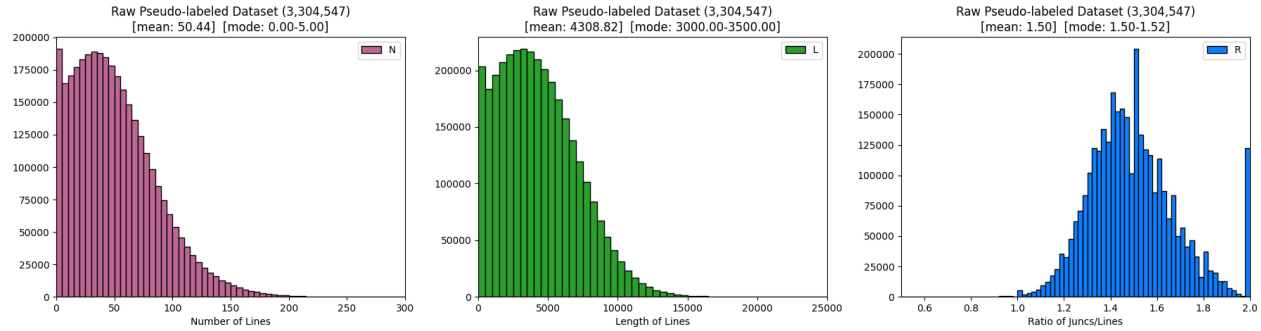
Algorithm 1 Avoid-Isolation Algorithm

function AVOIDISOLATION(s, K) ▷ s: a silhouette, K: number of max attempts (we use 500)
  if $rand(0, 1) > 0.8$ **then** ▷ By 20% chance, trying to create Hole Type 1 that contains no junction
    $mask \leftarrow$ randomly_place_hole_until_Hole_Type_1_found(s)
    if succeed to find Hole_Type_1 within K attempts **then**
      return $mask$
    end if
  end if
  $mask \leftarrow$ randomly_place_hole_until_Hole_Type_2_found(s)
  if fail to find Hole_Type_2 after K attempts **then**
    $n_{prev} \leftarrow \inf$
    $m_{prev} \leftarrow 0$
    for $k \leftarrow 1$ to K **do**
      $n \leftarrow$ number of line segments whose endpoints are both contained in the hole
      $m \leftarrow$ total length (= number of pixels) of line segments overlapping with the hole
      if $n \leq n_{prev}$ and $m > m_{prev}$ **then**
        $n_{prev} \leftarrow n$
        $m_{prev} \leftarrow m$
        $mask \leftarrow$ set_mask(s) ▷ After many attempts, it becomes (sub-)optimal Hole Type 3
      end if
    end for
  end if
  return $mask$
end function



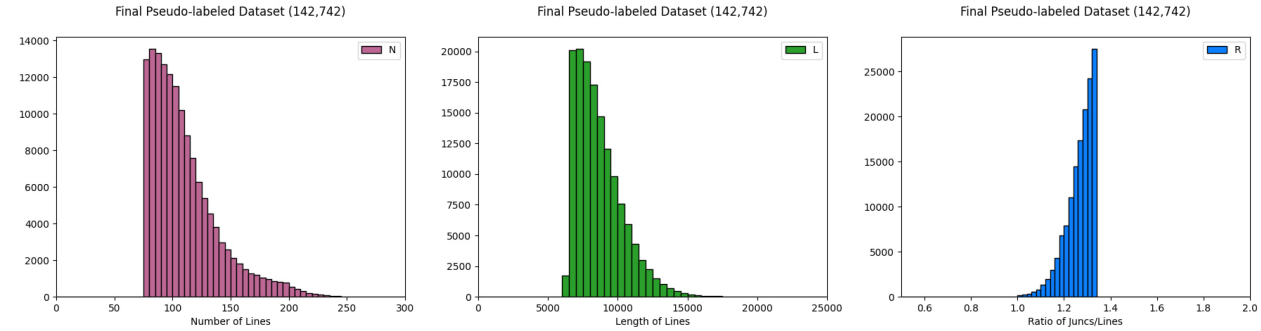
(a) Number of lines (b) Total length of all lines (c) Ratio of # junctions / # lines

Figure 5: **Histograms on three criteria for the ground truth labeled Wireframe training set [5].** Each histogram depicts the frequency of images over the entire Wireframe training set, with respect to the bins corresponding to either (a) number of lines, (b) total length of all lines, or (c) ratio of $\frac{\text{\# of Junctions}}{\text{\# of Lines}}$ in an image.



(a) Number of lines (b) Total length of all lines (c) Ratio of # junctions / # lines

Figure 6: **Histograms on three criteria for the *raw* pseudo-labeled dataset *before* applying the thresholds.** The distributions are either skewed as in (a) and (b), or irregular as in (c) due to outliers from inaccurate detection.



(a) Number of lines (b) Total length of all lines (c) Ratio of # junctions / # lines

Figure 7: **Histograms on three criteria for the *final* pseudo-labeled dataset *after* applying the thresholds.** Once the thresholds are applied, the new distributions look quite alike the ground truth distributions whose other half with respect to the mean value has been cut out.

4. Ground Truth and Pseudo Label Distributions on Three Criteria

- Number of lines in an image > 74.98
- Total length of all lines in an image > 6456.57
- The ratio $\frac{\text{\# of Junctions}}{\text{\# of Lines}}$ in an image < 1.34

The threshold values in these three criteria above were determined by inspecting the distributions of ground truth wireframe labels in each image of the original Wireframe training set [5], which follow quite ideal normal distributions. Histograms in Figure 5 depict those distributions on the three criteria.

We extracted the thresholds from the ground truth label distributions instead of pseudo label distributions, because it is hard to obtain accurate distributions from the pseudo labels as they may contain several outliers whose detection had failed badly. As shown in Figure 6, the *raw* pseudo-labeled dataset *before* applying the thresholds appear to be either skewed as in (a) and (b), or irregular as in (c) due to those outliers.

Assuming that ideal distributions of the pseudo labels would be similar to those of the ground truth labels on the three criteria, we applied the thresholds to the candidate pseudo-labeled examples, to pick only a subset of them more likely to have better detected and more dense labels. This effectively removes the outliers from the raw dataset: as shown in Figure 7, the new distributions of the final pseudo-labeled dataset after the thresholds are applied quite resemble the ground truth distributions in Figure 5 whose other half with respect to the mean value, corresponding to each of our threshold values, has been cut out.

5. Implementation Details

We implemented our approach using PyTorch, and trained each model on one V100 GPU. We keep the original hyper parameters in the basic frameworks as identical as possible. Here we describe training details of our approach applied to the HT-HAWP [12, 7] framework. When pseudo labeling is not applied, we trained the model on our modified Wireframe dataset 30 epochs in total. We progressively increase the hole size by +1% at every three epochs, such that 0.1-1%, 1-2%, . . . , 9-10%. We kept the default batch size 6 and learning rate 4e-4. When pseudo labeling is enabled, we first trained on the large pseudo-labeled data 10 epochs in total, while incrementing the hole size +1% at every epoch. We used a larger batch size 8, but kept the default learning rate 4e-4. We then fine-tuned on the modified Wireframe dataset 10 epochs in total, again +1% increment of the hole size at every epoch, while using a smaller batch size 6 and smaller learning rate 4e-5.

6. Extra Analysis and Studies

6.1. Ablation Studies on Conditional Training with Various Settings

				Wireframe Test Set [5]											
				10-30% Hole				0-10% Hole				without Hole			
	(a)	(b)	(c)	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H
HT-HAWP [12, 7]				35.64	38.54	41.8	65.5	51.58	55.50	55.1	80.1	63.26	67.12	61.3	**85.7**
	✓			44.79	48.72	**48.3**	74.1	58.04	62.18	58.5	83.0	62.78	66.59	60.7	85.4
	✓	✓		44.99	48.92	48.2	67.8	58.27	62.55	58.7	78.8	63.00	66.90	60.9	80.9
	✓	✓	✓	**45.11**	**48.94**	48.2	**74.4**	**58.59**	**62.85**	**59.2**	**83.7**	**63.31**	**67.19**	**61.4**	85.5

(a): Training with masked data with hole
(b): Progressive hole size increment along with epochs
(c): Placing a hole by avoiding isolated components
(a)+(b)+(c): Conditional training

Table 1: Ablation study on conditional training.

			Wireframe Test Set [5]											
			10-30% Hole				0-10% Hole				without Hole			
(a)	(b)	(c)	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H
(Trained on masked data with *"0.1-10% hole size"*)														
✓			44.79	48.72	48.3	**74.1**	58.04	62.18	58.5	83.0	62.78	66.59	60.7	**85.4**
✓	✓		44.99	**48.92**	48.2	67.8	**58.27**	**62.55**	**58.7**	78.8	**63.00**	**66.90**	**60.9**	80.9
(Trained on masked data with *"0.1-30% hole size"*)														
✓			**45.04**	48.89	**48.6**	74.0	57.63	61.79	**58.7**	**83.1**	62.45	66.28	**60.9**	**85.4**
✓	✓		**45.04**	48.91	**48.6**	70.7	57.67	62.02	58.5	80.1	62.20	66.17	60.6	82.7

(a): Training with masked data with hole
(b): Progressive hole size increment along with epochs

Table 2: Effect of larger hole size around *"0.1-30%"* instead of *"0.1-10%* on training with masked data. The basic framework is HT-HAWP [12, 7].

Table 1 shows an ablation study on our conditional training in various settings. We studied it with the HT-HAWP [12, 7] baseline on the Wireframe test set [5]. The ablations between "applying (a)" and "applying (a) (b)" show that progressively increasing the hole size is better than choosing a random size hole, since this scheme lets the training initially focus on ordinary detection and then, later on, adapt to gradually larger holes thus gives more time before facing more challenging tasks. The ablations between "applying (a) (b)" and "applying (a) (b) (c)" show that placing a hole in the image by carefully avoiding isolated components is more effective than placing it at an entirely random location, because a random placement may result in a scene component entirely hidden if the hole is large. This makes sense, as it would be irrational to expect the network to predict hidden structures without any visible clue left in the input image.

Table 2 studies an effect of larger hole size on training with masked data. Note that above 10%, it is hard to place such a large hole by strictly avoiding isolated components when generating masked data, thus we had to place the hole above 10% at an entirely random location. Since it is not possible to apply the avoid isolation scheme around 10-30% size, we chose to ablate only for (a) and (b) but except (c) avoid-isolation for this study. The numbers in this table show that training with a larger hole size around *"0.1-30%"* is not clearly beneficial at 10-30% hole inference than training with a smaller hole around 0.1-10%, and it works even worse at 0-10% hole inference and without-hole inference in general. Also, according to Table 1, applying the avoid isolation scheme is clearly beneficial. For these reasons, we concluded to use a 0.1-10% hole size range for creating our final conditional training data.

6.2. Applying GAN to Other Types of Information

	Wireframe Test Set [5]											
	10-30% Hole				0-10% Hole				without Hole			
	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H
HT-LCNN [15, 7] + Cond' training	42.08	45.69	47.6	70.8	55.21	59.29	58.5	78.7	60.20	64.11	60.9	81.2
+ junction heatmap GAN	29.06	31.79	31.0	48.9	37.55	41.04	37.8	59.0	43.73	47.63	40.9	65.5
+ line heatmap GAN	40.81	44.36	46.7	69.3	53.67	57.75	57.6	77.6	58.63	62.46	60.0	80.1
+ RGB GAN	**43.35**	**47.06**	**48.3**	**72.1**	**57.02**	**61.02**	**59.3**	**80.2**	**61.57**	**65.40**	**61.6**	**82.4**
HT-HAWP [12, 7] + Cond' training	45.11	48.94	**48.2**	74.4	58.59	62.85	59.2	83.7	63.31	67.19	61.4	85.5
+ full AFM GAN	45.41	49.02	47.9	73.2	58.96	63.03	58.9	82.6	63.52	67.24	61.1	84.7
+ distance map GAN	44.80	48.65	**48.2**	74.4	58.72	62.95	**59.3**	83.6	63.25	67.16	**61.5**	**85.7**
+ angular maps GAN	45.10	48.72	47.7	73.6	58.94	62.90	58.7	83.1	63.44	67.11	60.9	85.1
+ RGB GAN	**45.72**	**49.56**	48.0	**75.0**	**59.33**	**63.59**	59.1	**84.0**	**63.56**	**67.49**	61.2	85.6

Table 3: Ablation study on applying GAN to other components.

Table 3 shows how we concluded to apply GAN [3] to RGB generation instead of a junction, line, or attraction field map. The first study was made on HT-LCNN [15, 7] baseline, where combining junction map GAN or line map GAN significantly reduced the performance than without it. We suspect that the GAN is not suitable to estimate accurate sparse image maps for lines and junctions that are used to obtain coordinates information through direct conversion later. The second study was made on HT-HAWP [12, 7] that estimates a 4-D holistic Attraction Field Map (distance map and three more angular orientation maps) representing line segments. We applied GAN [3] to either all components of the AFM or only part of them. This worked quite well since these components of AFM are dense representation maps, but still, RGB GAN showed better numbers on structural metrics. We believe that learning to generate the scene contents through RGB GAN contributes to deeper scene understanding than indirectly going through the generation of structural maps.

6.3. Testing the Sole Effect of Pseudo Labeling

	Wireframe Test Set [5]											
	10-30% Hole				0-10% Hole				without Hole			
	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H
HAWP [12]	34.80	37.74	40.8	64.5	50.80	54.84	54.0	79.4	62.52	66.49	60.2	85.0
+ Pseudo labeling	**36.51**	**39.39**	**42.6**	**66.3**	**53.29**	**57.07**	**56.5**	**80.9**	**65.45**	**69.13**	**62.8**	**86.5**
HT-HAWP [12, 7]	35.64	38.54	41.8	65.5	51.58	55.50	55.1	80.1	63.26	67.12	61.3	85.7
+ Pseudo labeling	**36.63**	**39.55**	**42.9**	**66.5**	**53.30**	**57.21**	**56.5**	**81.2**	**65.64**	**69.40**	**63.1**	**86.5**
	York Urban Dataset [2]											
	10-30% Hole				0-10% Hole				without Hole			
	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H
HAWP [12]	15.79	17.48	22.5	46.3	21.61	23.79	28.6	57.3	26.15	28.54	31.6	61.3
+ Pseudo labeling	**16.37**	**18.22**	**24.2**	**47.6**	**22.43**	**24.77**	**30.0**	**58.0**	**26.75**	**29.18**	**32.8**	**61.6**
HT-HAWP [12, 7]	15.57	17.18	23.2	**46.8**	**21.45**	**23.63**	29.0	56.6	25.31	27.65	31.9	60.7
+ Pseudo labeling	**15.63**	**17.33**	**23.9**	46.4	21.20	23.37	**29.6**	**56.8**	**25.72**	**28.04**	**32.6**	**60.9**

Table 4: Effect of training on pseudo-labeled data.

In order to test the sole effect of pseudo labeling, we trained HAWP [12] and HT-HAWP [12, 7] baselines on our pseudo-labeled data without conditional masks. We applied 10 epochs on the pseudo data training as well as fine-tuning, same as our full approach works. As shown in Table 4, training on our large pseudo-labeled data helps to outperform the baseline model trained only on small ground truth labeled data in general, but these improvements are not as dramatic as our full approach cases, especially on the hole-robust performance gains. This again proves that the conditional training and RGB GAN in our full approach are important factors to conquer the hole occlusion problem.

7. F-Scores on Tables 1 and 2 in the Main Paper

	Wireframe Test Set [5]									York Urban Dataset [2]								
	10-30% Hole			0-10% Hole			without Hole			10-30% Hole			0-10% Hole			without Hole		
	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H
L-CNN [15]	42.71	44.87	67.1	52.19	54.61	74.4	59.12	61.31	76.9	27.00	28.36	54.6	31.99	33.53	60.1	35.42	36.92	61.9
HT-LCNN [15, 7]	43.74	45.81	65.3	53.40	55.72	76.2	60.53	62.71	79.0	28.10	29.76	53.1	33.59	35.34	58.6	37.14	38.91	60.5
F-Clip(HG2) [1]	44.89	47.54	69.3	54.85	57.86	77.6	62.04	64.94	80.7	28.43	30.26	56.3	33.60	35.44	62.4	37.50	39.28	64.3
F-Clip(HR) [1]	45.61	48.11	69.9	55.95	58.76	78.1	63.46	66.08	81.5	29.25	31.13	57.6	35.01	37.03	64.2	**38.90**	**40.82**	66.4
LETR [11]	45.28	48.45	71.4	57.08	60.55	80.4	62.59	66.12	**83.2**	28.88	31.40	57.7	34.04	37.10	64.6	37.52	40.50	**66.7**
HAWP [12]	44.94	47.04	69.1	55.29	57.71	77.4	62.63	64.87	80.6	28.03	29.67	55.6	33.86	35.56	62.7	37.88	39.61	65.3
HT-HAWP [12, 7]	45.28	47.35	70.0	55.69	58.03	78.3	63.06	65.19	81.6	27.91	29.46	56.5	33.57	35.31	63.1	37.36	39.08	65.1
HAWP + Our full approach	53.16	55.32	73.3	62.08	64.37	80.7	64.93	67.01	82.3	**32.74**	**34.40**	**59.6**	**37.43**	**39.18**	**64.7**	38.81	40.52	66.3
HT-HAWP + Our full approach	**53.48**	**55.63**	**75.1**	**62.57**	**64.90**	**81.8**	**65.44**	**67.58**	83.0	32.35	33.81	59.2	37.30	38.89	63.9	38.68	40.33	65.1

Table 5: **Effect of our full approach, evaluated on the Wireframe test set [5] and York Urban dataset [2], *with* and *without* hole.** sF^T (T: distance threshold) and F^H are F-score measurement for sAP^T and AP^H. See text for more details.

	Approach			Wireframe Test Set [5]									York Urban Dataset [2]								
				10-30% Hole			0-10% Hole			without Hole			10-30% Hole			0-10% Hole			without Hole		
	3.2	3.3	3.4	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H	sF^5	sF^{10}	F^H
L-CNN [15]				42.71	44.87	67.1	52.19	54.61	74.4	59.12	61.31	76.9	27.00	28.36	54.6	31.99	33.53	60.1	35.42	36.92	61.9
	✓			46.39	48.54	67.7	54.38	56.76	75.2	58.28	60.50	**77.4**	30.48	31.81	52.6	**35.43**	**37.15**	58.1	**37.24**	**38.80**	59.6
	✓	✓		**47.59**	**49.84**	**69.5**	**55.87**	**58.21**	**75.4**	**59.48**	**61.68**	77.1	**30.65**	**32.34**	**56.5**	34.87	36.51	**60.9**	36.66	38.28	**62.3**
HT-LCNN [15, 7]				43.74	45.81	65.3	53.40	55.72	76.2	60.53	62.71	79.0	28.10	29.76	53.1	33.59	35.34	58.6	37.14	38.91	60.5
	✓			48.20	50.24	70.3	56.43	58.67	76.1	60.15	62.28	77.5	30.35	31.98	53.6	34.60	36.23	57.7	**36.38**	**38.05**	58.8
	✓	✓		**49.23**	**51.41**	**71.7**	**57.95**	**60.18**	**77.8**	**61.35**	**63.51**	**79.2**	**30.62**	**32.08**	**58.4**	**34.68**	**36.43**	**62.0**	36.29	38.02	**62.9**
(a): HAWP [12]				44.94	47.04	69.1	55.29	57.71	77.4	62.63	64.87	80.6	28.03	29.67	55.6	33.86	35.56	62.7	37.88	39.61	65.3
(b):	✓			50.19	52.32	71.5	58.75	61.08	78.6	62.34	64.45	80.5	30.78	32.20	57.3	35.34	37.05	62.6	37.24	38.87	64.3
(c):	✓	✓		50.30	52.59	72.1	59.18	61.55	78.9	62.64	64.85	80.6	31.71	33.21	58.8	36.41	38.22	64.4	38.31	39.94	65.4
(d):	✓		✓	52.87	54.83	72.7	61.63	63.87	80.0	**64.95**	66.88	81.7	32.52	34.11	58.1	37.37	**39.31**	63.9	**39.00**	**40.76**	65.1
(e):	✓	✓	✓	**53.16**	**55.32**	**73.3**	**62.08**	**64.37**	**80.7**	64.93	**67.01**	**82.3**	**32.74**	**34.40**	**59.6**	**37.43**	39.18	**64.7**	38.81	40.52	**66.3**
(f): HT-HAWP [12, 7]				45.28	47.35	70.0	55.69	58.03	78.3	63.06	65.19	81.6	27.91	29.46	56.5	33.57	35.31	63.1	37.36	39.08	65.1
(g):	✓			50.87	53.06	72.9	59.52	61.83	79.6	63.11	65.21	81.3	30.67	32.26	59.0	35.70	37.31	63.6	37.41	39.10	65.0
(h):	✓	✓		51.71	53.97	73.9	60.64	63.02	80.4	63.83	66.02	82.0	30.97	32.54	59.3	35.92	37.70	63.9	37.30	38.98	64.8
(i):	✓		✓	52.37	54.27	73.5	61.15	63.41	80.7	64.25	66.33	82.1	31.07	32.52	**59.6**	35.74	37.34	**64.0**	37.46	38.98	**65.2**
(j):	✓	✓	✓	**53.48**	**55.63**	**75.1**	**62.57**	**64.90**	**81.8**	**65.44**	**67.58**	**83.0**	**32.35**	**33.81**	59.2	**37.30**	**38.89**	63.9	**38.68**	**40.33**	65.1
F-Clip(HG2) [1]				44.89	47.54	69.3	54.85	57.86	77.6	62.04	64.94	80.7	28.43	30.26	56.3	33.60	35.44	62.4	37.50	39.28	64.3
	✓			50.47	53.31	72.1	58.94	62.07	79.0	62.28	65.23	80.6	31.47	33.50	58.1	36.41	38.45	63.4	38.07	39.91	**64.8**
	✓	✓		**51.15**	**54.05**	**72.7**	**59.66**	**62.94**	**79.9**	**62.85**	**65.75**	**81.1**	**31.97**	**33.85**	**58.5**	**36.53**	**38.69**	**63.6**	**38.25**	**40.23**	**64.8**
F-Clip(HR) [1]				45.61	48.11	69.9	55.95	58.76	78.1	63.46	66.08	81.5	29.25	31.13	57.6	35.01	37.03	64.2	38.90	40.82	**66.4**
	✓			**52.61**	**55.47**	**73.5**	**61.52**	**64.35**	**80.6**	**64.40**	**66.96**	**82.0**	**33.43**	**35.30**	**59.0**	**38.03**	**40.36**	**64.5**	**39.69**	**41.70**	65.6

Table 6: **Applying our approach to various recent frameworks.** Approach 3.2: *Conditional Training*, Approach 3.3: *RGB GAN*, and Approach 3.4: *Pseudo Labeling*. Best metrics within each framework group are marked in **bold** font. Our approach can be applied to various recent frameworks sharing the *stacked HourGlass* [9] backbone, and is consistently effective both *without* and *with* hole in general.

Table 5 and Table 6 show F-Scores on Tables 1 and 2 in the Main Paper, respectively.

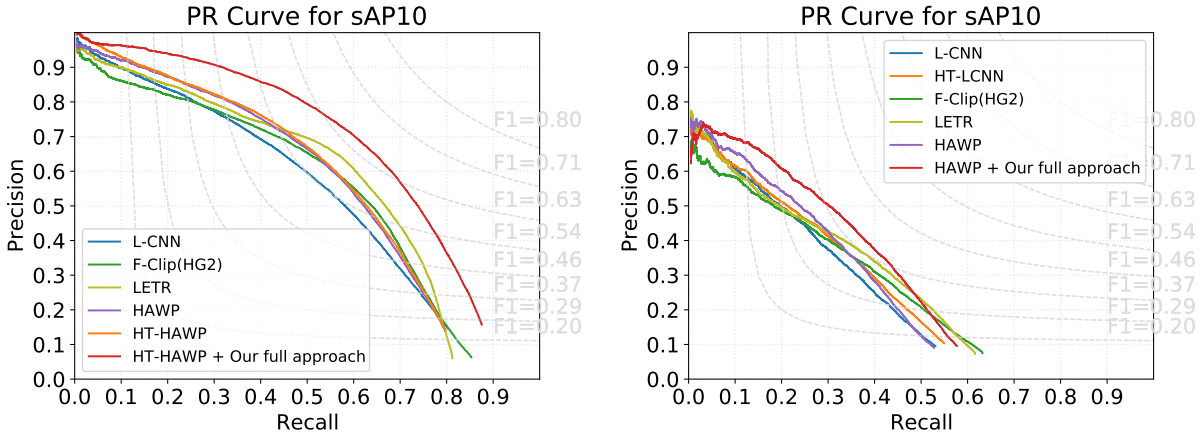
Table 5 shows an almost identical trend as Table 1 in the Main Paper. On the Wireframe test set, our full model (HT-HAWP + Our full approach) shows the best hole-robust performance at all metrics with hole, and the best numbers at sF^5 and sF^{10} metrics without hole, and the second best at F^H without hole. On the York Urban dataset, our full model (HAWP + Our full approach) shows the best hole-robust performance at all metrics with hole. Excluding F-Clip(HR) and LETR from Table 5, our full model (HAWP + Our full approach) on the York Urban dataset presents the best numbers at all metrics regardless of with hole or without hole. Also, note that any of our full models consistently improves the ordinary detection performance without hole against the corresponding basic framework.

F-scores in Table 6 show that at least one of the models enabling our approach performs better than its basic model at all cases. It proves the effectiveness of applying at least one of our approaches, 3.2 (Conditional Training), 3.3 (RGB GAN) and 3.4 (Pseudo Labeling). Also, our model enabling all three approaches performs the best in general.

8. Extra Precision-Recall Curves

8.1. PR Curves for sAP^{10} on Tests in Table 1 of the Main Paper

PR curves for sAP^{10} on the tests with 0-10% hole in Table 1 of the Main Paper are shown in Figure 8. The red curve from our full model shows a significant hole-robust performance improvement for both test sets.



(a) Wireframe test set [5], **0-10% hole**

(b) York Urban [2], **0-10% hole**

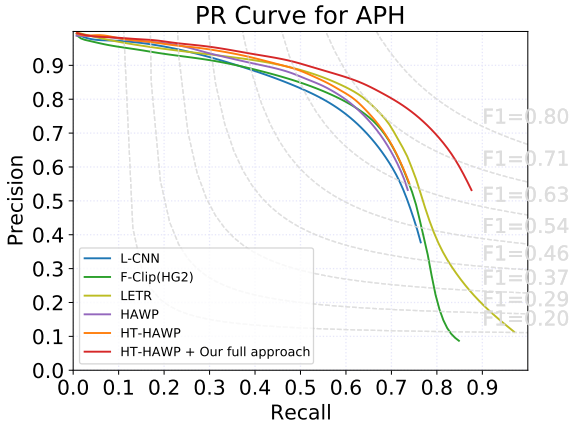
Figure 8: PR curves for sAP^{10} on the tests with *0-10%* hole in Table 1 of the Main Paper.

8.2. PR Curves for AP^H on Tests in Table 1 of the Main Paper

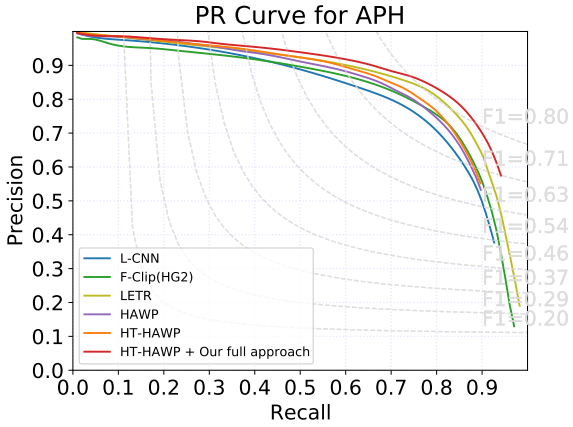
Figure 9 and Figure 10 show Precision-Recall curves drawn for AP^H on the tests *with* and *without* hole in Table 1 of the Main Paper.

The gap between our full model's curve and existing models' curves is visibly large at 10-30% hole in both test sets as shown in Figure 9 (a) and (c). The gap is still obvious in case of the Wireframe test set with 0-10% hole in Figure 9 (b). In case of the York Urban dataset with 0-10% hole, it is not easy to tell from Figure 9 (d), but the AP^H number is better than all others compared in the graph (see Table 1 in the Main Paper), and also, its corresponding PR curves for sAP^{10} in Figure 8 (b) show a clear superiority of our full model.

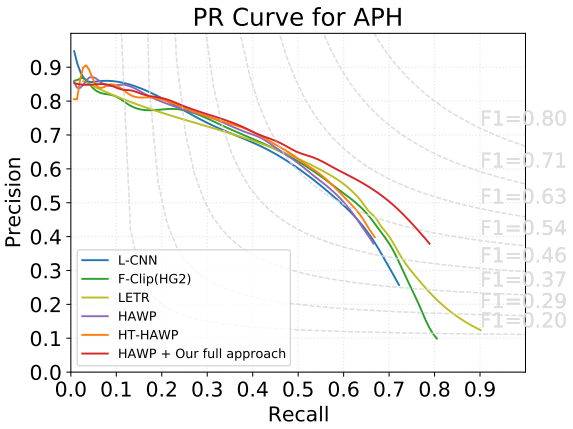
In the cases without hole in Figure 10, it is again hard to tell from the graphs intuitively, but the AP^H number is better than all others compared in either of these graphs (see Table 1 in the Main Paper).



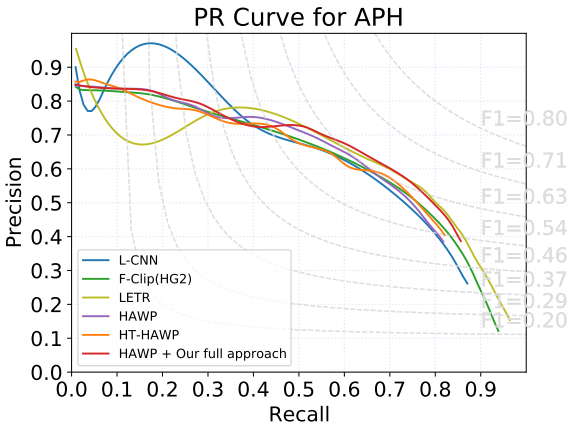
(a) Wireframe test set [5], **10-30% hole**



(b) Wireframe test set [5], **0-10% hole**

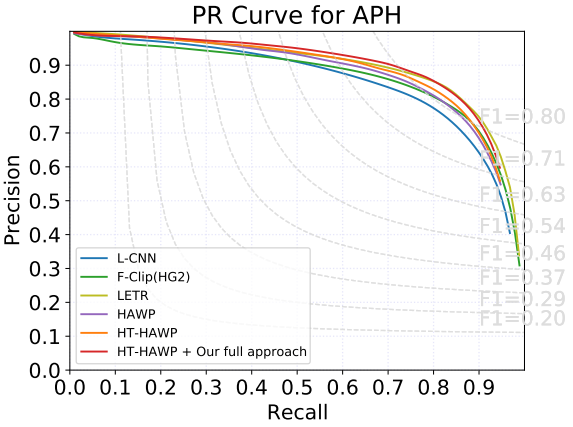


(c) York Urban [2], **10-30% hole**

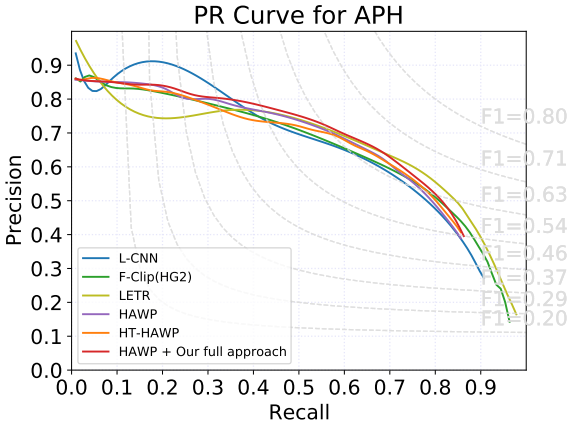


(d) York Urban [2], **0-10% hole**

Figure 9: PR curves for AP^H on the tests *with* hole in Table 1 of the Main Paper.



(a) Wireframe test set [5], ***without*** **hole**



(b) York Urban [2], ***without*** **hole**

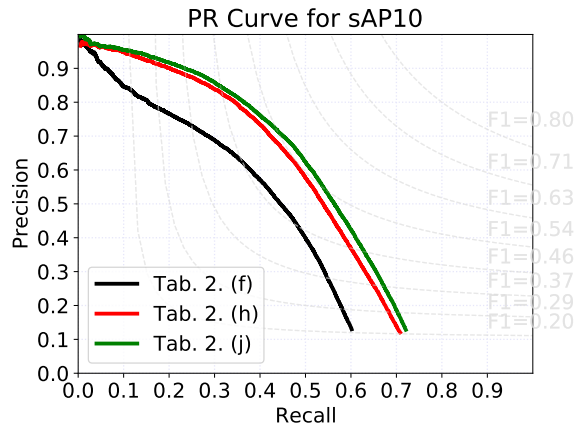
Figure 10: PR curves for AP^H on the tests *without* hole in Table 1 of the Main Paper.

8.3. PR Curves for sAP^{10} on Tests in Table 2 of the Main Paper

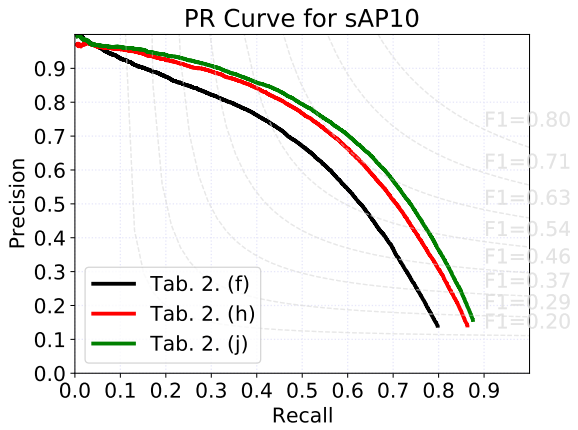
Figure 11 and Figure 12 show Precision-Recall curves drawn for the structural quality metric sAP^{10} with respect to (a),(c),(e) and (f),(h),(j) cases tested in Table 2 of the Main Paper. Figure 11 proves that compared to baselines (black curves), our approach yet without pseudo labeling (red curves) is already significantly effective in improving hole-robustness, which is even further improved with pseudo labeling (green curves). Figure 12 proves that our approach helps slightly enhance ordinary detection as well, and with pseudo labeling, the gain becomes more distinct.

8.4. PR Curves for AP^{H} on Tests in Table 2 of the Main Paper

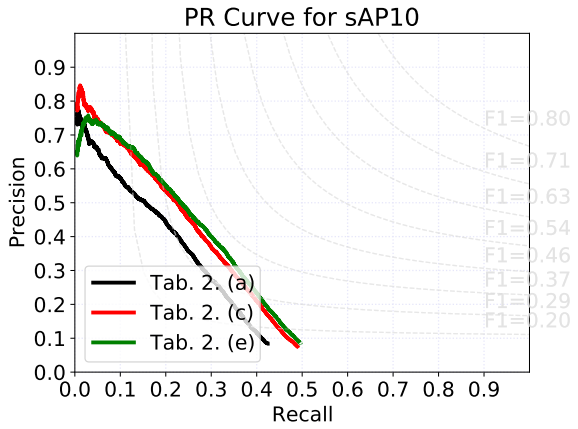
Figure 13 and Figure 14 show Precision-Recall curves drawn for AP^{H} with respect to (a),(c),(e) and (f),(h),(j) cases tested in Table 2 of the Main Paper. From these curves, we can again conclude as similarly as the PR-curves drawn for sAP^{10} in Section 8.3.



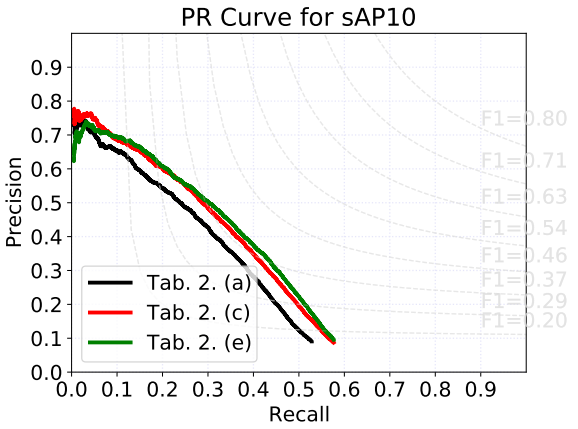
(a) Wireframe test set [5], **10-30% hole**



(b) Wireframe test set [5], **0-10% hole**

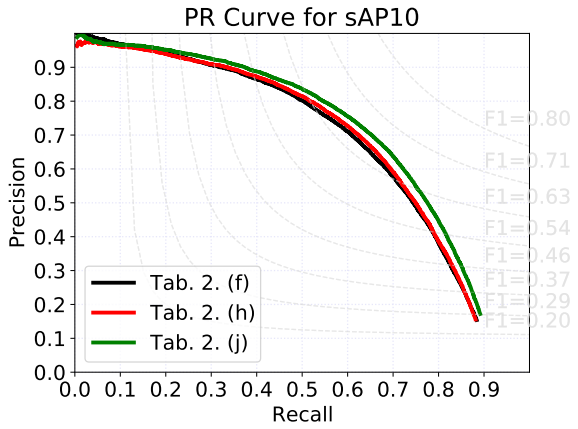


(c) York Urban [2], **10-30% hole**

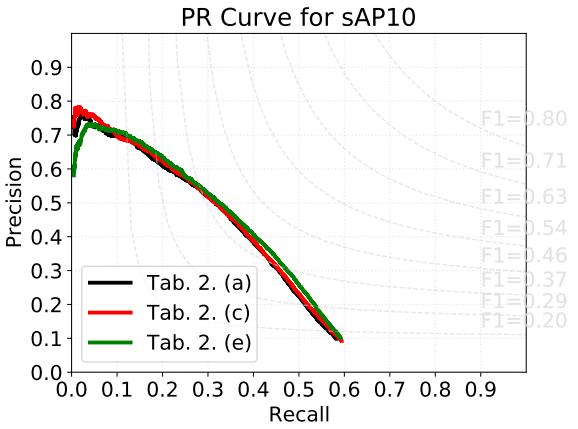


(d) York Urban [2], **0-10% hole**

Figure 11: PR curves for sAP^{10} on the tests *with* hole in Table 2 of the Main Paper.

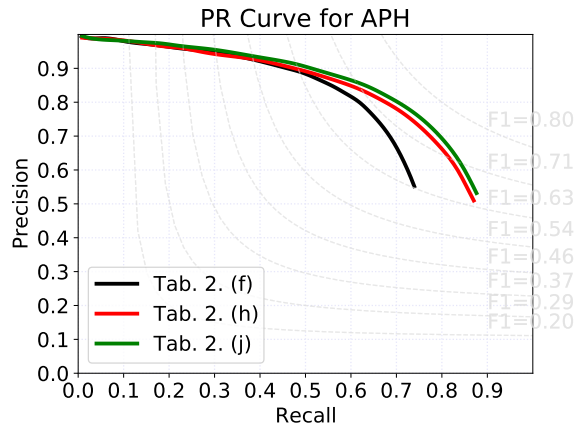


(a) Wireframe test set [5], ***without*** **hole**

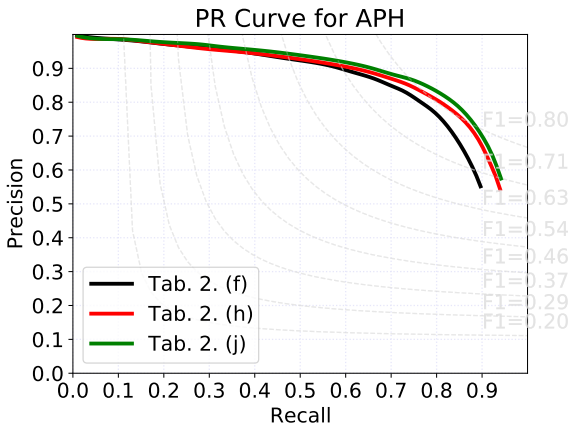


(b) York Urban [2], ***without*** **hole**

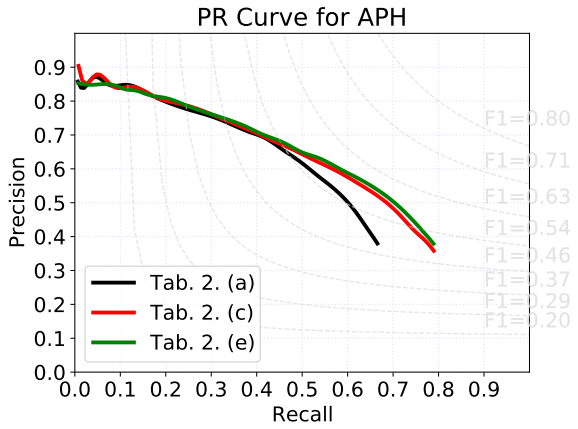
Figure 12: PR curves for sAP^{10} on the tests *without* hole in Table 2 of the Main Paper.



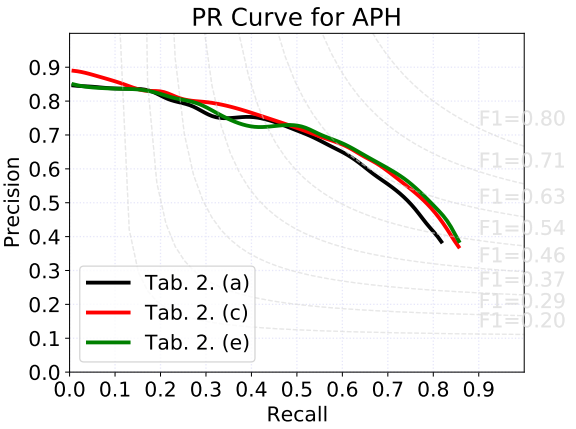
(a) Wireframe test set [5], **10-30% hole**



(b) Wireframe test set [5], **0-10% hole**

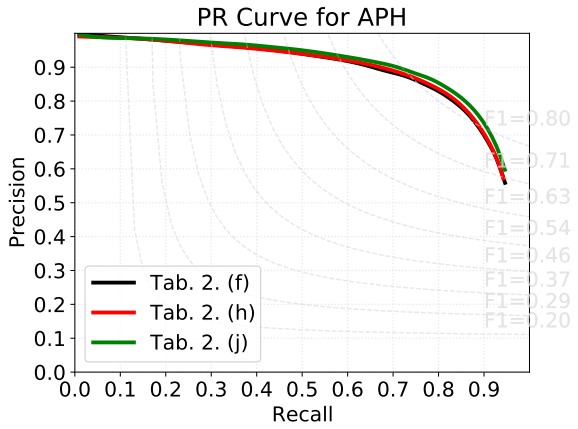


(c) York Urban [2], **10-30% hole**

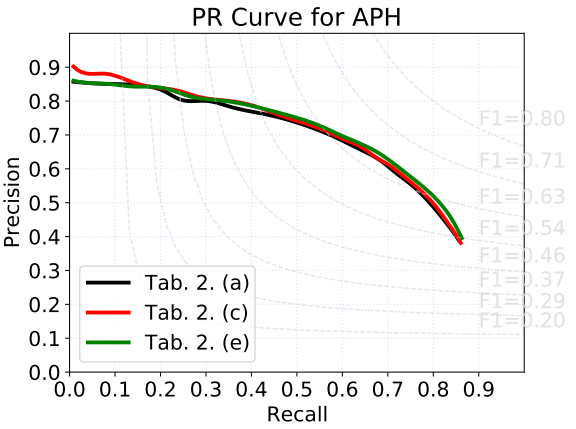


(d) York Urban [2], **0-10% hole**

Figure 13: PR curves for AP^H on the tests *with* hole in Table 2 of the Main Paper.



(a) Wireframe test set [5], ***without*** **hole**



(b) York Urban [2], ***without*** **hole**

Figure 14: PR curves for AP^H on the tests *without* hole in Table 2 of the Main Paper.

9. Initializing the Training with a Pretrained Ordinary Detection Model

	Wireframe Test Set [5]											
	10-30% Hole				0-10% Hole				without Hole			
	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H
(a): HAWP + 3.2 + 3.3	44.17	48.11	**47.1**	**73.1**	57.99	**62.35**	**58.2**	**82.9**	62.60	66.60	**60.5**	**85.1**
+ Init w/ Pretrained	**44.45**	**48.26**	**47.1**	72.5	**58.17**	62.34	58.1	82.6	**62.89**	**66.69**	60.4	84.8
(b): HT-HAWP + 3.2 + 3.3	45.72	49.56	**48.0**	**75.0**	59.33	63.59	**59.1**	**84.0**	63.56	67.49	**61.2**	**85.6**
+ Init w/ Pretrained	**46.24**	**49.89**	**48.0**	74.9	**60.05**	**64.12**	**59.1**	83.9	**64.19**	**67.90**	**61.2**	**85.6**
	York Urban Dataset [2]											
	10-30% Hole				0-10% Hole				without Hole			
	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H
(c): HAWP + 3.2 + 3.3	**20.30**	**22.34**	**25.8**	**53.0**	**24.90**	**27.35**	**31.2**	**60.6**	**26.69**	**29.08**	**32.4**	**62.3**
+ Init w/ Pretrained	18.47	20.32	24.7	51.2	23.18	25.34	29.3	58.7	24.62	26.76	30.5	60.4
(d): HT-HAWP + 3.2 + 3.3	19.44	21.39	24.6	52.8	**24.35**	**26.62**	29.6	**59.6**	**25.91**	**28.16**	30.5	**61.0**
+ Init w/ Pretrained	**19.91**	**21.92**	**25.6**	**53.0**	24.27	26.60	**29.8**	58.3	25.74	27.99	**30.7**	59.1

Table 7: Initializing the training with a pretrained ordinary detection model. For the tested models we combined our approach 3.2: *'Conditional Training'* and 3.3: *'RGB GAN'* on top of two basic frameworks HAWP [12] and HT-HAWP [12, 7]. Default training uses random initialization, which worked meaningfully better at (c). We see no clear difference at (a), (b) and (d).

We train a modified backbone network through our approach by initializing the kernel weights with random numbers as stated in [4] (i.e., *kaiming_uniform*). In this section, we test an effect of initializing our modified backbone with pretrained weights from the basic model made for ordinary detection without hole. Since the modified backbone network and the original backbone network are architecturally different in some parts (see Section 2), we copy the pretrained weights only for those matching layers while initializing the rest through default *kaiming_uniform*.

Table 7 shows the results on this experiment. On the Wireframe test set [5], initializing with the pretrained weights was slightly more helpful than full random initialization, but the difference was marginal. On the York Urban dataset [2], training with random initialization was much better in case of (c): HAWP + 3.2 + 3.3, while it is still unclear which initialization method is better in case of (d): HT-HAWP + 3.2 + 3.3. This suggests that initialization with pretrained weights for the baseline model does not have a clear benefit compared to random initialization, or could make it even worse. We suspect that the reason is, 1) our models are not sensitive to initialization methods, and 2) our modified backbone learns different information through the RGB GAN approach, which adds a generative role to the backbone network as part of a scene contents generator, unlike the baseline backbone dedicated only to learn ordinary detection through sole supervision.

10. Robustness to a Dim or Over Lit Condition

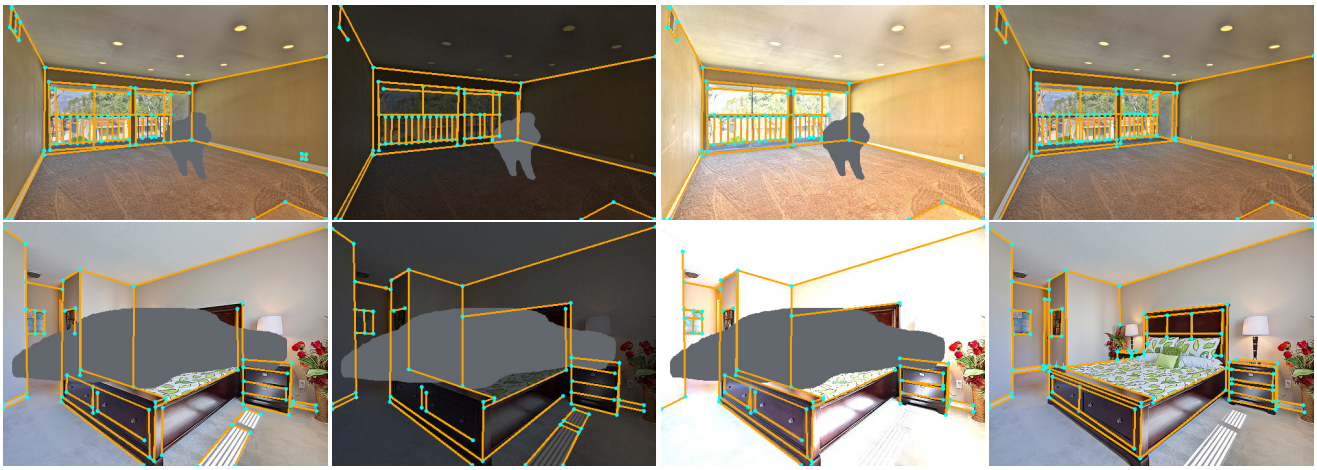
In this section we test the robustness of our models to a dim or over lit condition that we may rarely but do encounter while capturing a photo in real life. We simulated images captured under the dim or over lit condition by adjusting the original source images in the Wireframe Test Set [5] as follows:

1. Linearizing the camera response (assuming gamma 2.2)
2. Randomly scaling the image value (assuming the original image value is within $[0, 255]$)
   - for dim lit: scale by a random $s \in [\frac{1}{16.0}, \frac{1}{8.0}]$ and truncate below 0.
   - for over lit: scale by a random $s \in [3.0, 3.3]$ and truncate above 255.
3. (dim lit only) Adding Poisson noise (shot noise), with peak $\lambda = 8.0$
4. Reapplying the camera response (gamma 2.2)

	(a)	(b)	(c)	Wireframe Test Set [5]											
				10-30% Hole				0-10% Hole				without Hole			
				sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H
Baseline HAWP	✓			34.80	37.74	40.8	64.5	50.80	54.84	54.0	79.4	62.52	66.49	60.2	85.0
		✓		29.89	33.02	35.7	59.4	44.05	48.25	47.8	74.1	54.74	59.08	54.1	79.9
			✓	27.96	30.78	34.0	55.5	41.15	45.05	45.0	69.0	50.80	54.82	50.4	74.1
HAWP + Ours full	✓			**47.59**	**51.50**	**49.6**	**74.9**	**61.83**	**65.98**	**61.0**	**85.0**	**65.98**	**69.76**	**63.0**	**86.9**
		✓		41.19	45.23	44.4	69.9	54.09	58.53	54.9	80.2	58.32	62.47	57.0	82.1
			✓	38.43	42.14	41.0	64.5	50.14	54.30	50.5	73.5	53.74	57.59	52.3	75.5
Baseline HT-HAWP	✓			35.64	38.54	41.8	65.5	51.58	55.50	55.1	80.1	63.26	67.12	61.3	85.7
		✓		29.80	32.85	35.6	59.6	43.76	48.02	47.8	74.4	54.64	59.01	54.3	80.2
			✓	28.45	31.18	34.5	56.2	41.43	45.16	45.7	69.8	51.03	54.81	51.2	74.6
HT-HAWP + Ours full	✓			**48.02**	**51.82**	**49.9**	**76.6**	**62.21**	**66.31**	**61.3**	**85.5**	**66.19**	**69.92**	**63.2**	**87.0**
		✓		41.52	45.51	44.3	71.3	54.24	58.92	54.7	80.4	58.33	62.60	56.8	82.1
			✓	38.85	42.40	41.3	66.1	50.52	54.59	50.9	74.3	53.95	57.75	52.6	75.8

(a): normal lit
(b): dim lit
(c): over lit

Table 8: Testing robustness to dim or over lit conditions. We compared the basic frameworks HAWP [12] and HT-HAWP [12, 7] with those modified by applying our full approach.



(a) Normal lit (b) Dim lit (c) Over lit (d) Ground Truth

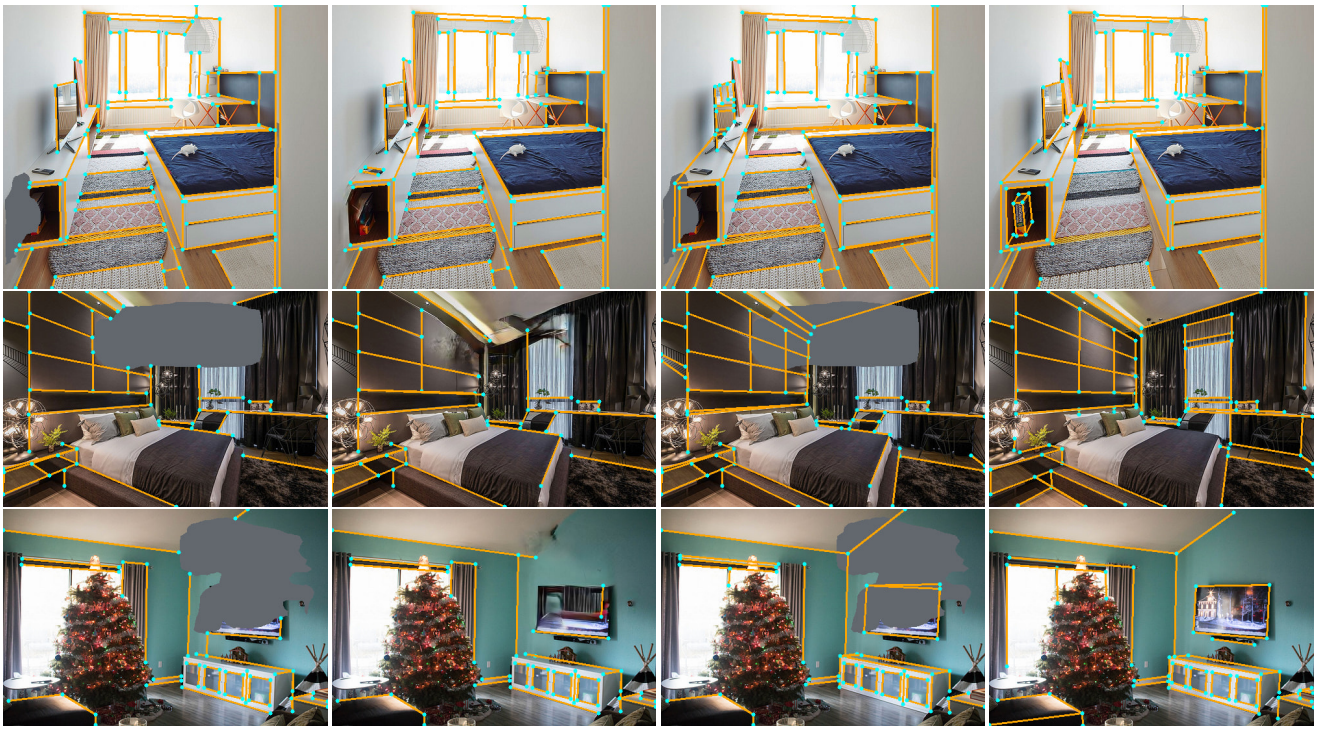
Figure 15: Results on dim and over lit conditions. HT-HAWP [12, 7] with our full approach was applied to detect wireframes visualized in (a), (b) and (c).

Table 8 show results on this experiment. We compared the basic frameworks HAWP [12] and HT-HAWP [12, 7] with those modified by applying our full approach. We found that the performance drop is more prominent in the over lit case than the dim lit case, since the over lit scenario introduced stronger and larger saturated (white) regions than the dim lit one. Here, a model with our full approach always performed far better than its corresponding basic model on the same input condition, either without or with hole. Especially, with 10-30% hole, the model with our full approach applied to challenging *over lit* images performed even better than the basic model applied to *normal* images. We suspect that our large scale training on the pseudo-labeled data derived from diverse images may have helped improve the robustness of the model on unnatural lighting conditions. Figure 15 visualizes simulated images under a dim or over lit condition and detection results on them.

11. Testing Ordinary Detection Models on Inpainted Images

	Wireframe Test Set [5]								York Urban Dataset [2]							
	10-30% Hole				0-10% Hole				10-30% Hole				0-10% Hole			
	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H	sAP^5	sAP^{10}	mAP^J	AP^H
(Detecting on images with hole)																
L-CNN [15]	32.83	35.78	40.4	61.3	47.73	51.68	53.2	75.3	14.65	16.16	21.9	44.6	20.23	22.15	27.5	54.6
HT-LCNN [15, 7]	33.56	36.48	41.4	57.1	48.89	52.79	54.2	76.6	15.84	17.48	23.5	40.4	21.47	23.62	29.3	50.1
F-Clip(HG2) [1]	33.51	36.68	/	65.3	49.46	53.87	/	79.5	15.86	17.54	/	48.6	22.02	24.07	/	58.6
HAWP [12]	34.80	37.74	40.8	64.5	50.80	54.84	54.0	79.4	15.79	17.48	22.5	46.3	21.61	23.79	28.6	57.3
HT-HAWP [12, 7]	35.64	38.54	41.8	65.5	51.58	55.50	55.1	80.1	15.57	17.18	23.2	46.8	21.45	23.63	29.0	56.6
(Detecting on **inpainted** images by applying [13])																
L-CNN [15]	39.01	42.84	45.0	70.5	53.30	57.70	56.4	78.4	17.02	19.07	24.0	50.9	22.80	24.92	29.1	56.8
HT-LCNN [15, 7]	40.32	44.01	46.3	71.1	54.70	59.04	57.6	79.5	18.27	20.43	25.4	46.8	24.00	26.36	31.0	51.9
F-Clip(HG2) [1]	40.87	45.04	/	75.2	55.50	60.39	/	82.6	18.81	21.07	/	**56.9**	25.05	27.43	/	**60.7**
HAWP [12]	41.82	45.67	45.7	73.7	56.61	61.10	57.1	82.8	18.53	20.61	24.5	53.7	24.32	26.70	30.2	60.1
HT-HAWP [12, 7]	42.52	46.32	46.7	74.3	57.36	61.75	58.4	83.5	18.01	20.19	25.4	54.8	23.60	26.00	30.6	58.7
(Detecting on images with hole)																
HAWP + Our full approach	47.59	51.50	49.6	74.9	61.83	65.98	61.0	85.0	**20.81**	**23.01**	**27.0**	53.4	**25.62**	**28.01**	**31.7**	60.2
HT-HAWP + Our full approach	**48.02**	**51.82**	**49.9**	**76.6**	**62.21**	**66.31**	**61.3**	**85.5**	19.96	21.92	25.4	52.9	24.72	26.94	29.8	57.6

Table 9: Testing existing works on images whose holes had been inpainted by applying a popular method Deepfillv2 [13]. (Note that we excluded F-Clip(HR) [1] and LETR [11] from the comparison as they are fundamentally very different from the other frameworks thus hard to be compared in parallel.)



(a) HT-HAWP with hole (b) HT-HAWP on inpainted input (c) Ours with hole (d) Ground Truth

Figure 16: Applying an ordinary detection model, HT-HAWP [12, 7], to images inpainted by using Deepfillv2 [13]. Artifacts in the inpainted regions may still prevent proper inference of line segments as shown in (b). Ours: our full approach using HT-HAWP as the basic framework. Hole size: 0-10% in the first row, 10-30% in the second and third rows.

In Table 9 we tested existing ordinary wireframe detection models on images whose holes had been inpainted by applying a popular inpainting method Deepfillv2 [13]. This does help improve the performance as shown in the second group of the table. However, the inpainting result may not be perfect inside large holes such as those at 10-30%, which suffers from crooked structure, disturbing artifacts or blurry completion. This abnormality may prevent proper inference of line segments across the inpainted regions as shown in Figure 16 (b). We found that in general, our full approach using HAWP or our full approach using HT-HAWP applied to non-inpainted images with hole still performs better than those existing models applied to inpainted images.

12. More Qualitative Results

In this section, we show a more number of qualitative results and real-world examples that could not be put in the Main Paper due to space.

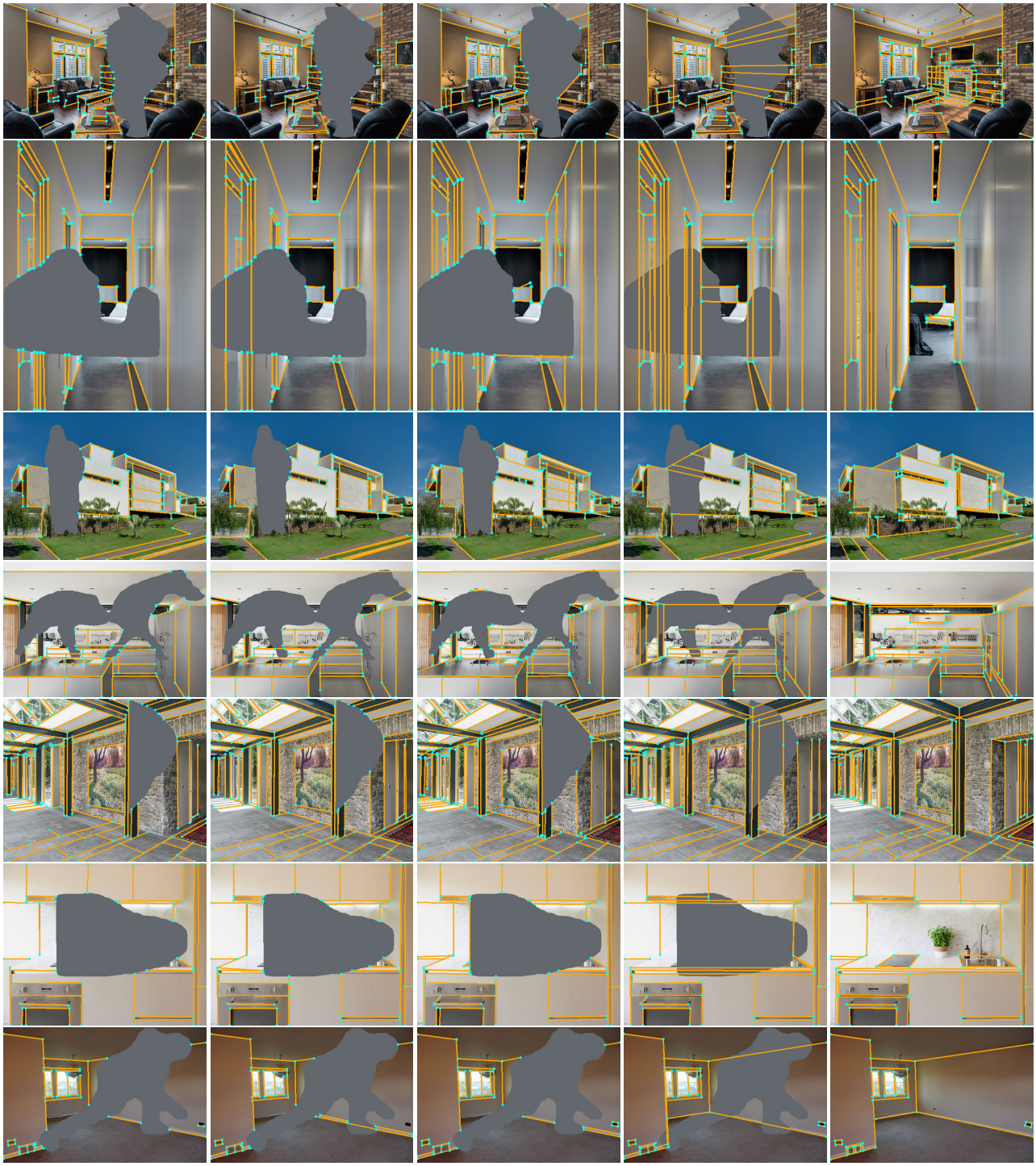
12.1. Results on Standard Test Sets with and without Hole

We show extra results on the Wireframe test set [5] *with* 10-30% hole in Figure 17, and *with* 0-10% hole in Figure 18. In Figure 19, we show extra results on York Urban dataset [2] *with* 10-30% and 0-10% hole. We show extra results on both test sets *without* hole in Figure 20.

These results are also from the same methods compared in Figures 6 and 7 of the Main Paper.

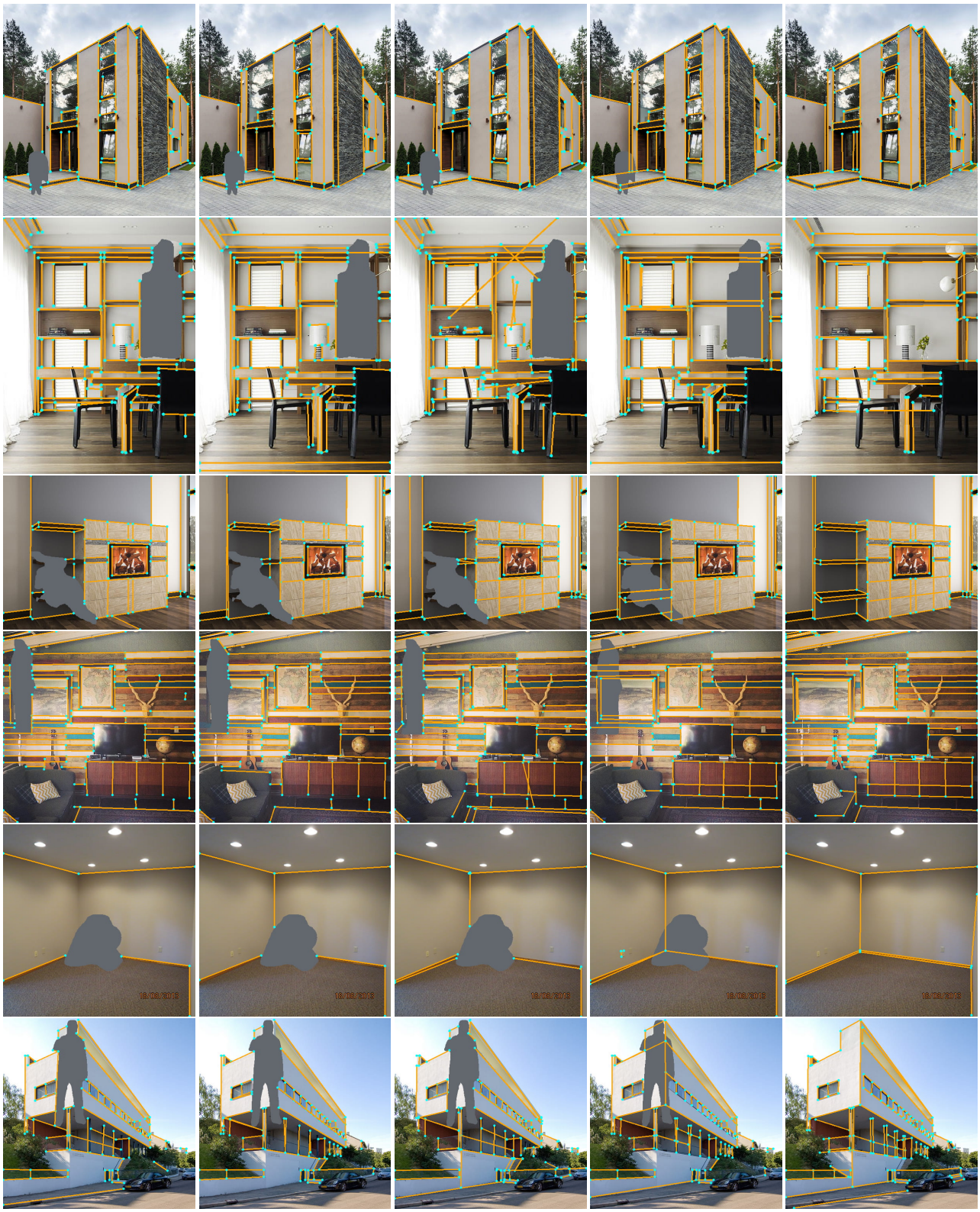
12.2. Real-world Examples with Occlusion by Foreground Objects

In Figures 21, 22, 23 and 24, we demonstrate additional qualitative results on real-world examples with obvious occlusion by foreground objects. The example images were sampled and cropped (512×512) from the Places365 [14] Validation set. We apply existing works HT-LCNN [15, 7], HT-HAWP [12, 7], and F-Clip(HR) [1] to an input image as is. We use Detectron2 [10] panoptic segmentation ('*things*' only) to indicate the occluded region by a foreground object in the image, and apply our hole-robust detection full model to it.



(a) HT-LCNN [15, 7] (b) HT-HAWP [12, 7] (c) F-Clip(HR) [1] (d) Our full approach using HT-HAWP [12, 7] (e) Ground Truth

Figure 17: Results on the **Wireframe test set** [5] ***with* 10-30% hole.**

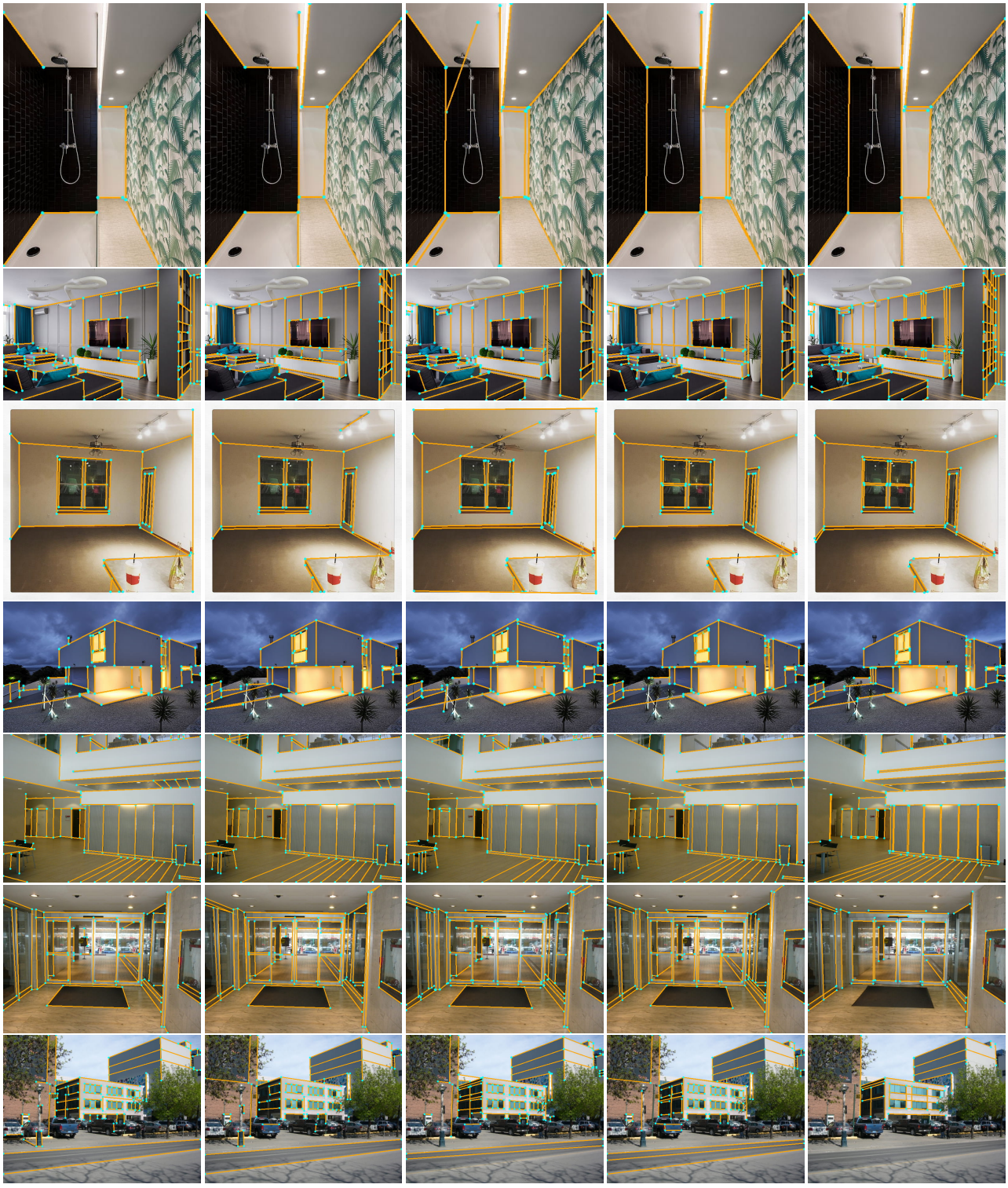


(a) HT-LCNN [15, 7] (b) HT-HAWP [12, 7] (c) F-Clip(HR) [1] (d) Our full approach using HT-HAWP [12, 7] (e) Ground Truth

Figure 18: Results on the **Wireframe test set** [5] ***with* 0-10% hole.**

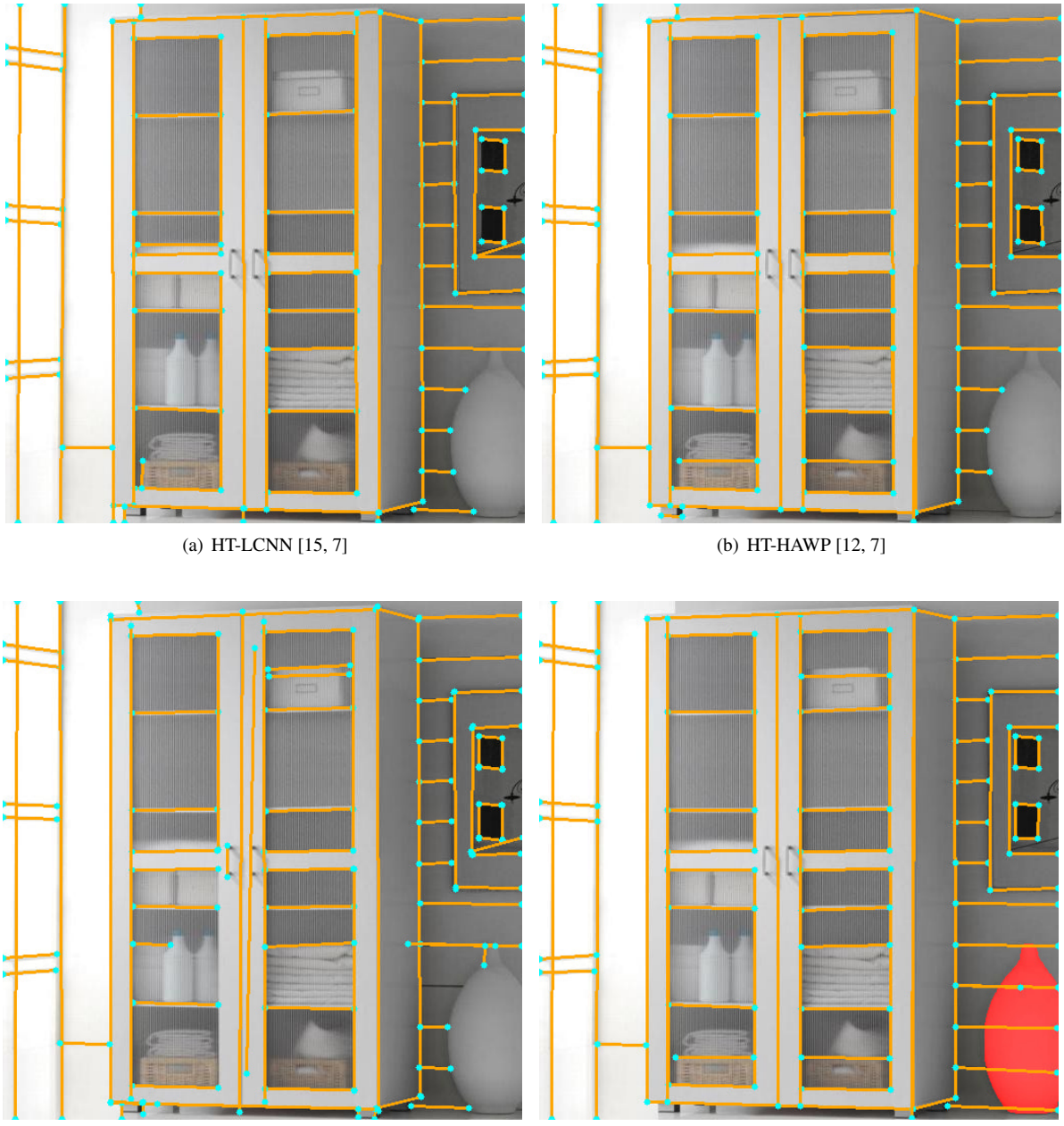
(a) HT-LCNN [15, 7] (b) HT-HAWP [12, 7] (c) F-Clip(HR) [1] (d) Our full approach using HT-HAWP [12, 7] (e) Ground Truth

Figure 19: Results on the **York Urban dataset** [2] ***with* hole.** (**10-30%** from 1st to 4th rows, **0-10%** from 5th to 8th rows)



(a) HT-LCNN [15, 7] (b) HT-HAWP [12, 7] (c) F-Clip(HR) [1] (d) Our full approach using HT-HAWP [12, 7] (e) Ground Truth

Figure 20: Results on the **Wireframe test set** [5] (1st to 4th rows) and **York Urban dataset** [2] (5th to 7th rows) ***without* hole.**



(a) HT-LCNN [15, 7]

(b) HT-HAWP [12, 7]

(c) F-Clip(HR) [1]

(d) Our full approach using HT-HAWP [12, 7]

Figure 21: Results on real examples with occlusion by a foreground object (white vase on the bottom right corner). (a)-(c) failed to detect wireframe vectors behind the foreground object. (d) The foreground object region from panoptic segmentation [10] is shown in transparent red color. Using this segmented region, we provided a pair of an image with hole and a mask map (not shown above) to our full model. The hidden structures are well detected as shown in the visualized image.

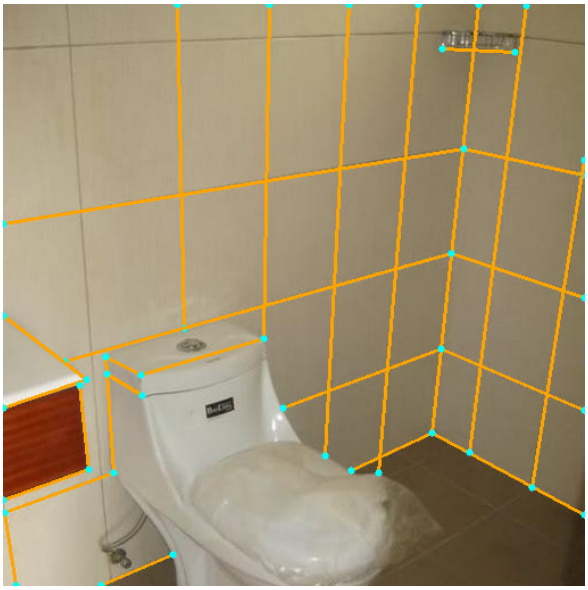
(a) HT-LCNN [15, 7]

(b) HT-HAWP [12, 7]

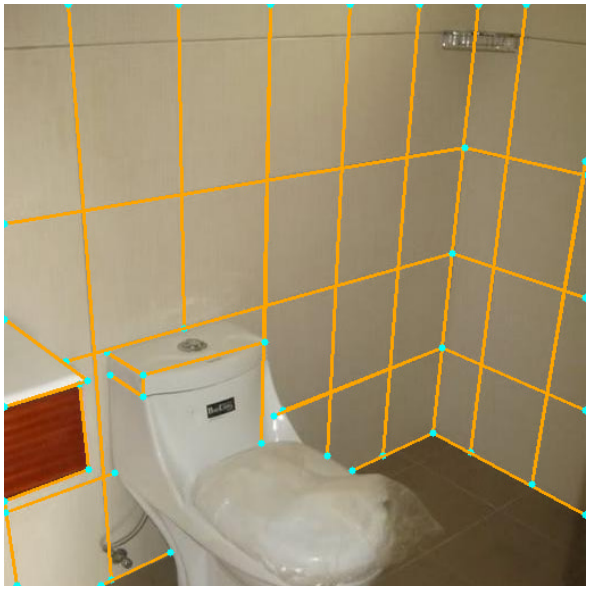
(c) F-Clip(HR) [1]

(d) Our full approach using HT-HAWP [12, 7]

Figure 22: Results on real examples with occlusion by foreground objects (cup and beer can). (a)-(c) do not well detect wireframe vectors behind the foreground objects. (d) The foreground object regions from panoptic segmentation [10] is shown in transparent red color. Using these segmented regions, we provided a pair of an image with hole and a mask map (not shown above) to our full model. The hidden structures are well detected as shown in the visualized image.



(a) HT-LCNN [15, 7]

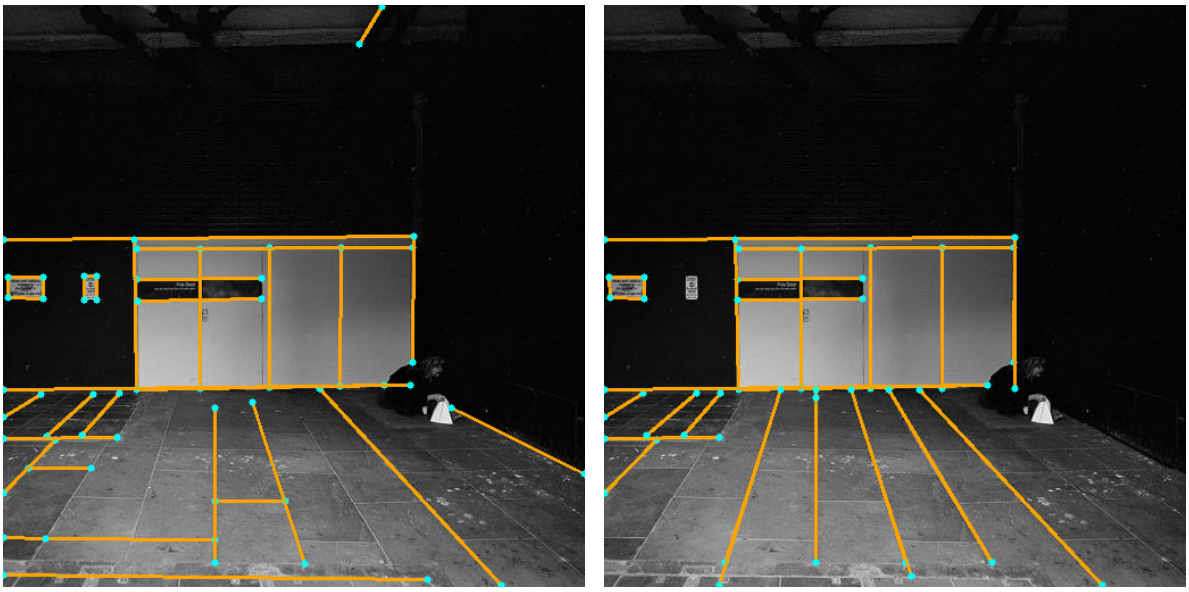


(b) HT-HAWP [12, 7]

(c) F-Clip(HR) [1]

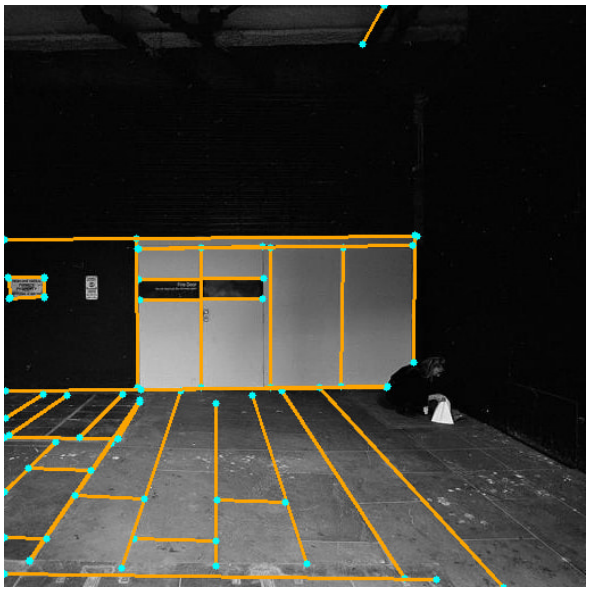
(d) Our full approach using HT-HAWP [12, 7]

Figure 23: Results on real examples with occlusion by a large foreground object (toilet). (a)-(c) failed to detect wireframe vectors behind the foreground object. (d) The foreground object region from panoptic segmentation [10] is indicated in transparent red color. Using this segmented region, we provided a pair of an image with hole and a mask map (not shown above) to our full model. The hidden structures are well detected as shown in the visualized image.

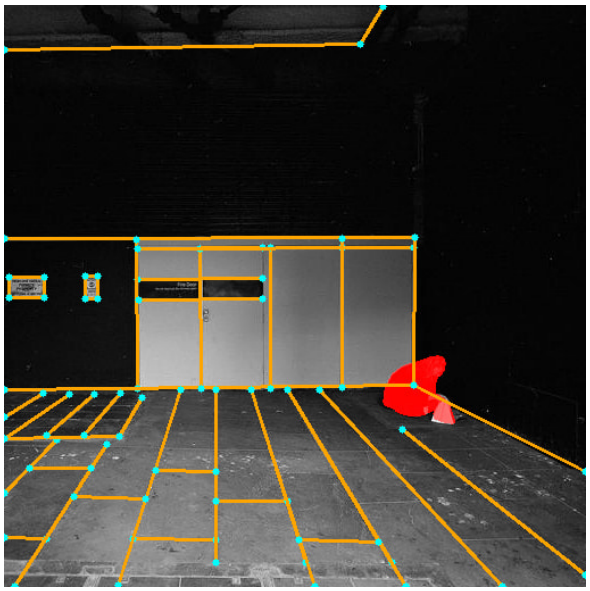


(a) HT-LCNN [15, 7]

(b) HT-HAWP [12, 7]



(c) F-Clip(HR) [1]



(d) Our full approach using HT-HAWP [12, 7]

Figure 24: Results on real examples with occlusion by a foreground object (crouching person in black clothes, quite invisible due to the black background). (a)-(c) do not well detect wireframe vectors behind the foreground object. (d) The foreground object region from panoptic segmentation [10] is indicated in transparent red color. Using this segmented region, we provided a pair of an image with hole and a mask map (not shown above) to our full model. The hidden structures are well detected as shown in the visualized image.

References

[1] Xili Dai, Xiaojun Yuan, Haigang Gong, and Yi Ma. Fully convolutional line parsing. *arXiv preprint arXiv:2104.11207*, 2021.

[2] Patrick Denis, James H Elder, and Francisco J Estrada. Efficient edge-based methods for estimating Manhattan frames in urban imagery. In *European Conference on Computer Vision (ECCV)*, pages 197–210. Springer, 2008.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[5] Kun Huang, Yifan Wang, Zihan Zhou, Tianjiao Ding, Shenghua Gao, and Yi Ma. Learning to parse wireframes in images of man-made environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 626–635, 2018.

[6] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017.

[7] Yancong Lin, Silvia L Pintea, and Jan C van Gemert. Deep Hough-transform line priors. In *European Conference on Computer Vision (ECCV)*, pages 323–340. Springer, 2020.

[8] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[9] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision (ECCV)*, pages 483–499. Springer, 2016.

[10] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. 2019. *URL https://github.com/facebookresearch/detectron2*, 2(3), 2019.

[11] Yifan Xu, Weijian Xu, David Cheung, and Zhuowen Tu. Line segment detection using transformers without edges. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4257–4266, 2021.

[12] Nan Xue, Tianfu Wu, Song Bai, Fudong Wang, Gui-Song Xia, Liangpei Zhang, and Philip HS Torr. Holistically-attracted wireframe parsing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2788–2797, 2020.

[13] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[14] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1452–1464, 2017.

[15] Yichao Zhou, Haozhi Qi, and Yi Ma. End-to-end wireframe parsing. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 962–971, 2019.