

Multi-Plane Program Induction with 3D Box Priors

Yikai Li^{1,2*} Jiayuan Mao^{1*} Xiuming Zhang¹

William T. Freeman^{1,3} Joshua B. Tenenbaum¹ Noah Snavely³ Jiajun Wu⁴

¹MIT CSAIL

²Shanghai Jiao Tong University

³Google Research

⁴Stanford University

<http://bpi.csail.mit.edu>

This supplementary document is organized as the following. First, we revisit the related work in Appendix A. Second, in Appendix B, we describe four-step inference algorithm we use to infer box programs for inner views in detail. Third, in Appendix C, we show the mathematical details of how to reconstruct the 3D positions and surface normal vectors of different planes based on the plane segmentation for a box’s inner views (see Appendix D for outer views). Fourth, in Appendix D, we present the Box Program Induction (BPI) applied to *outer views* of boxes. Next, in Appendix E, we discuss the implementation details of BPI, including the domain-specific language. Finally, we present more qualitative and quantitative results on box program induction, plane segmentation, image inpainting, and image extrapolation in Appendix F.

A. Related Work

Visual program induction. Computer graphics researchers have used procedural modeling (mostly top-down) for representing 3D shapes [18, 33] and indoor scenes [35, 19, 29]. With advances in deep networks, some methods have paired top-down procedural modeling with bottom-up recognition networks. Such hybrid models have been applied to hand-drawn sketches [11], scenes with simple 2D or 3D geometric primitives [31, 24], and markup code [9, 5]. The high-dimensional nature of procedural programs poses significant challenges to the search process; hence, even guided by neural networks, these works focus only on synthetic images in constrained domains. SPIRAL [12] and its follow-up SPIRAL++ [26] use reinforcement learning to discover latent “doodles” that are later used to compose the image. Their models work on in-the-wild images, but cannot be directly employed in tasks involving explicit reasoning, such as image manipulation and analogy making, due to the lack of program-like, interpretable representations.

In the past year, Young et al. [40] and Mao et al. [25] integrated formal programs into deep generative networks to represent natural images, and later applied the hybrid representation to image editing. Li et al. [20] extended these

models by jointly inferring perspective effects. All these models, however, assume a single plane in an image, despite the fact that most images contain multiple planes such as floor and ceiling. Our BPI moves beyond the single-plane assumption by leveraging box priors.

Image manipulation. Image manipulation, in particular image inpainting, is a long-standing problem in graphics and vision. Pixel-based [1, 3] and patch-based methods [10, 4, 14] achieve impressive results for inpainting regions that requires only local, textural information. They fail on cases requiring high-level, structural, or semantic information beyond textures. Image Melding [7] extends patch-based methods by allowing additional geometric and photometric transformations on patches, but ignores global consistency among patches. Huang et al. [16] also use perspective correction to assist patch-based inpainting, but rely on vanishing points detected by other methods. In contrast, BPI segments planes and estimates their normals based on the global regularity of images.

Advances in deep networks have led to impressive inpainting algorithms that integrate information beyond local pixels or patches [17, 39, 41, 23, 42, 46, 38, 37, 28]. In particular, deep models that learn from a single image (a.k.a., internal learning) produce high-quality results for image manipulation tasks such as inpainting, resizing, and expansion [32, 46, 30]. BPI also operates on a single image, simultaneously preserving the 3D structure and regularity during image manipulation.

B. Box Program Induction for *Inner views*

In this section, we present the details of the box program induction for inner views of boxes, following Sec. 2.3 of our main paper.

Step 1: Visual cue detection. Following the *box prior*, all inner views of a box contain a single *vanishing point* and four *intersection lines* that are the intersection between the two walls, the ceiling, and the floor in 3D. These intersection

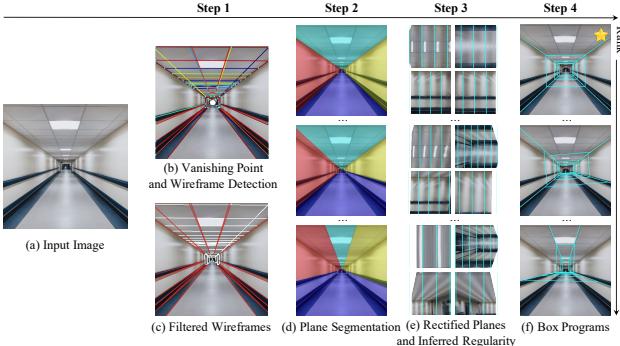


Figure 1: Our Box Program Induction finds the best-fit program that describes the input image (a). It first detects the vanishing point and wireframe segments (b), followed by a filtering step (c). It then constructs a set of candidate plane segmentation maps (d). Given each plane segmentation, it rectifies each plane and infers its regularity structure (e). We rank all candidate box programs by their fitness (f); the starred candidate is the best.

lines will be projected onto the image plane as four lines that intersect at the vanishing point. We use this property to constrain the candidate plane segmentation. Leveraging vanishing point information for inferring box structures of scenes has also been studied in [15].

Given an input image (Fig. 1a), we apply NeurVPS [44] to detect the vanishing point and L-CNN [45] to extract wireframes in the image. We use the most confident prediction of NeurVPS as the vanishing point $vp \in \mathbb{R}^2$, which is a 2D coordinate on the image plane. Each wireframe segment is represented as a segment AB on the image, from (x_i^A, y_i^A) to (x_i^B, y_i^B) , as illustrated in Fig. 1b. Next, we filter out wireframe segments whose length is smaller than a threshold δ_1 or whose extension does not cross a neighbourhood centered at vp with radius δ_2 . The remaining wireframe segments, denoted by the set WF , are illustrated in Fig. 1c.

Step 2: Plane segmentation. We then enumerate all combinations of four wireframe segments from WF . As these wireframe segments $s_i = [(x_i^A, y_i^A), (x_i^B, y_i^B)]$, $i = 1, 2, 3, 4$ may not intersect at a single point, we compute a new vanishing point vp^* that minimizes $\sum_i dist(vp^*, s_i)$, where $dist$ is the distance between the point vp^* and the line containing segment s_i . Next, we connect the new vanishing point vp^* and the farther end of each s_i to get four rays. These four rays partition the image into four parts, which we treat as the segmentation of four planes, as illustrated in Fig. 1d.

Step 3: Plane rectification and regularity inference. Fixing the camera at the world origin, pointing in the $+z$ direction, we then compute the 3D position and surface normal of each plane. Since the distance between camera and the corridor is coupled with the focal length of the camera,

here we use a fixed focal length of $f = 35\text{mm}$ ^{*}. Based on these assumptions, the four rays can *unambiguously* determine the 3D positions and surface normals of four planes. The proof can be found in the supplemental material.

Based on the inferred surface normal, we can *rectify* each plane, yielding a set of rectified images $\{J_1, J_2, \dots, J_4\}$. For each rectified plane, we search for the best plane program that describes it, based on the fitness function $F = -\sum_{a,b} [\|J_i[c(a,b)] - J_i[c(a+1,b)]\|_2^2 + \|J_i[c(a,b)] - J_i[c(a,b+1)]\|_2^2]$. The inferred plane regularity structures are shown in Fig. 1e.

Step 4: Box program ranking. We sum up the fitness score for four planes in each candidate segmentation as the overall program fitness. We use this score to rank all candidate segmentations, and choose the program with the highest fitness to describe the entire image.

C. Plane Reconstruction from Segmentation in Inner Views

In an inner view of a box, we use the plane segmentation of the input image to determine the 3D positions and surface normal vectors of four planes. The plane segmentation is represented as four rays starting from the detected vanishing point. Our reconstruction assumes a pinhole camera model with no lens distortion, as illustrated in Fig. 2a.

We start by defining the 3D coordinate system. We define the position of the camera pinhole O as origin of the coordinate system. We also define the optical axis of the camera (i.e., the ray from the center of the image plane Π' to O) as the $+z$ axis.

V' denotes the vanishing point shown on the image plane. Four rays starting from V' will intersect with the image boundary at four points $\{I'_k \mid k = 1, 2, 3, 4\}$. The line segments $\{V'I'_k \mid k = 1, 2, 3, 4\}$, namely the intersection line segments, are 2D projections of the intersection lines in 3D, between four planes. We also denote these 3D intersections lines as $\{I_k E_k \mid k = 1, 2, 3, 4\}$, where I_k is the corresponding 3D projection of I'_k , and E_k is the 3D projection of an arbitrary point on the 2D line segment $V'I'_k$. Thus, all lines $\{I'_k I_k \mid k = 1, 2, 3, 4\}$ should intersect at the optical center O .

As can be seen in Fig. 2a, the focal length (i.e. the distance between the image center and the optical center O) is correlated with the distance between I_k and O (i.e., the camera-to-subject distance)[†]. Moreover, the aspect ratio of

^{*}Following common practice, we also fix other camera intrinsic properties: optical center to $(0, 0)$, skew factor to 0, and pixel aspect ratio to 1.

[†]Formally, we need three planes that are perpendicular to each other to determine the focal length based on the perspective effect. However, we have only two such planes here. See Liebowitz et al. [21] for a detailed discussion.

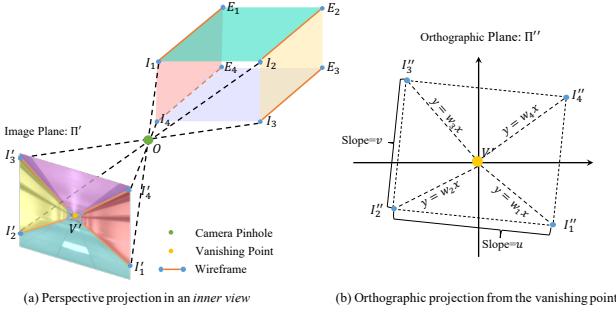


Figure 2: Illustration of (a) the perspective projection in an inner view of the box (image upside down to be consistent with the projection), and (b) the orthographic projection centered at the vanishing point.

the image sensor is correlated with the ratio between the sizes of four planes (i.e., the “aspect ratio” of the box). Thus, it is impossible to fully determine the focal length and the camera aspect ratio from this single image. Given this ambiguity, we assume the focal length to be 35mm and the aspect ratio to be 1, and then optimize for the equivalent distance between I_k and O .

To this end, we first consider the following property of vanishing point: the line $V'O$ should be parallel with all 3D intersection lines $I_k E_k$. Thus, we perform an *orthographical projection* of the inner view using a new optical axis $V'O$. This leads to a new image Π'' , as illustrated in Fig. 2b. The line segment OI_k will also be projected onto Π'' as a line segment OI''_k , and four intersection lines $I_k E_k$ will become four points on Π'' . Determining the distance between O and I_k is equivalent to determining the distance between I''_k and V' on the new image plane Π'' .

Unfortunately, one can not fully determine the distance between I''_k and V' , but only the ratio between $V'I''_1$ and $V'I''_k$, $k = 2, 3, 4$, even if we have assumed the focal length and the aspect ratio. This is because this camera-to-subject distance is also correlated with the actual size of the box. Intuitively, a box that is close and small may look identical in the image as another box that is far but big. Thus, we will manually set the distance between V' and I''_1 to be 1 meter. It is important to note that, although we have manually set the focal length, the aspect ratio, and the camera-to-subject distance, these values of these parameters will not affect the plane rectification results. Moreover, they also have no influence on downstream tasks such as image inpainting.

With the position of I''_1 , we then determine the positions of I''_k , $k = 2, 3, 4$, relative to the position of I''_1 . We use the following *box prior*: four planes of the box are either perpendicular or parallel to each other. Thus, on the orthographic image plane Π'' , the quadrilateral $I''_1I''_2I''_3I''_4$ should be a rectangle.

Now we use the 2D coordinates defined on the plane Π''

and centered at the vanishing point V' . Denote the coordinates of I''_k on Π'' by (x_k, y_k) , the slope of $I''_1I''_2$ by u , and the slope of $I''_2I''_3$ by v . Also denote the slope of VI''_k by w_k , $k = 1, 2, 3, 4$, as illustrated in Fig. 2b. We have the following equation system:

$$\begin{cases} y_k = w_k x_k; k = 1, 2, 3, 4 & (I''_k \text{ lies on the ray } VI''_k.) \\ y_1 - y_2 = u(x_1 - x_2) & (\text{definition.}) \\ y_2 - y_3 = v(x_2 - x_3) & (\text{definition.}) \\ y_3 - y_4 = u(x_3 - x_4) & (I''_1I''_2 \text{ is parallel to } I''_3I''_4.) \\ y_4 - y_1 = v(x_4 - x_1) & (I''_2I''_3 \text{ is parallel to } I''_4I''_1.) \\ x_1^2 + y_1^2 = 1 & (\text{camera assumption.}) \\ uv = -1 & (I''_1I''_2 \text{ is perpendicular to } I''_2I''_3) \end{cases}$$

This equation system allows us to solve for u and v unambiguously. In fact, there exists a closed-form solution to the values of u and v , which is independent of (x_k, y_k) . After determining u and v , we can further compute the positions of I''_k and thus the 3D positions of I_k , $k = 1, 2, 3, 4$.

During inference, we first compute the orthographic projection Π'' based on the detected vanishing point and the focal length. Next, by fixing the position of I''_1 , we solve for the other I''_k , $k = 2, 3, 4$ on the orthographic image. Finally, we project the solution back to the 3D space and determine the 3D positions and surface normal vectors for individual planes.

D. Box Program Induction for Outer Views

In this section, we present the box program induction for *outer views* of boxes. The whole process is almost identical to the *inner view* case, except that for an outer view, we only need to consider two planes (e.g., two side planes of a building), with the other planes being either non-visible or foreshortened severely. The full search process consists of four steps. First, we use pre-trained neural networks to detect the 2D wireframe line segments (but no vanishing points) from the image. We also filter out wireframe segments that are too short in length. Next, we generate a set of candidate plane segmentation maps by partitioning the image based on the detected wireframe segments. Then, for each segmented plane, we seek the program that best describes the regularity on each plane. Finally, we rank all candidate plane segmentations by the fitness score (defined in the main text).

Step 1: Visual cue detection. Following the *box prior*, an outer view of a box contains only two planes. Therefore, there will be only one intersection line between these two planes (e.g., two walls of a building). We use L-CNN [45] to extract wireframes in the image. Next, we filter out wireframe segments whose length is smaller than a threshold δ_1 . The remaining wireframe segments are denoted by the set

WF. Note that unlike the inner view case, we do not use vanishing point detection for outer views.

Step 2: Plane segmentation. Next, we then extend every wireframe segment to a line. Each line will partition the input image into two parts, which we treat as the candidate plane segmentation of the image.

Step 3: Plane rectification and regularity inference. Since we only have two planes, we cannot use the plane segmentation to fully reconstruct the positions and surface normal vectors of different planes. We run the Perspective Plane Program Induction (P3I) algorithm [20] on each plane to jointly infer the surface normal of each plane and its regularity structure.

Step 4: Box program ranking. Identical to the inner view case, we sum up the fitness score for two planes in each candidate segmentation as the overall program fitness. We use this score to rank all candidate segmentations, and pick the program with the highest fitness to describe the entire image.

E. Implementation Details

Table 1 specifies the domain-specific language (DSL) we use in box program.

When filtering wireframes by length, we set $\delta_1 = 0.1 \times \min(w, h)$, where w, h are the width and height of the input image, respectively. Radius used to filter wireframes toward the detected vanishing point is $\delta_2 = 0.01 \times \min(w, h)$. In the plane rectification step, we rectify the plane region to a 200×200 image, on which we infer the regularity. We assume that objects repeat at least 3 times on each plane.

F. Additional Results

F.1. Qualitative results.

We supplement more results on box program induction (Fig. 6). Our model can be applied to less constrained images by allowing the user to specify the regular region. It also works for images where not all planes of a box exhibit regular patterns. Fig. 6 (i) shows example results. In (a), we run BPI on a user-specified region (the orange bounding box). BPI outputs a reasonable program even though the left plane is curved. We also show that our model can be applied to a broader class of images than buildings, such as the bamboo forest in (b), and the scene with irregular planes in (c).

F.2. Plane Segmentation

Because BPI develops a 3D understanding of the scene in the process of inferring the box program under perspective effects, an immediate application is single-image plane segmentation. Note that BPI performs this task by jointly inferring the perspective effects and the pattern regularity, in contrast to supervised learning approaches that train models

with direct supervision, such as [22]. The core intuition is that the correct plane segmentation leads to the program that best describes each segmented plane in terms of repeated visual elements.

Baselines. We compare our method with two baselines: Huang et al. [16] and PlaneRCNN [22]. Huang et al. [16] first uses VLFeat [34] to detect vanishing points and line segments. It then generates a soft partition of the image into multiple parts by the support lines. PlaneRCNN is a learning-based method for plane segmentation. It uses a Mask-RCNN-like pipeline [13] and is trained on ScanNet [6].

Metrics. The output of each model is a set of masks indicating the segmented planes. We compare these with the ground-truth masks by computing the best match between two sets of masks, where the matching metric is the intersection over union (IoU). We then report the average IoU of all predicted masks. For corridor images, we exclude pixels in the far plane region during evaluation. Because we wish to use the plane segmentation to aid in image manipulation tasks such as image inpainting, we evaluate all methods on both original images (CrdO, BldO) and corrupted images (CrdC, BldC).

Results. Fig. 3 and Table 2 show that BPI consistently outperforms the baselines on both corridor and building images. The baselines fail to detect planes when they contain complex structures and patterns. We also provide more qualitative results for plane segmentation in Fig. 7.

F.3. Image Inpainting

The inferred box programs support 3D-aware and regularity-preserving image manipulation, because they provide information on both what the perspective effects are and how the visual element repeats. To test BPI’s performance on such tasks, we generate a dataset of corrupted images by randomly masking out regions of images from our two datasets.

Metrics. We use four metrics: pixel-level \mathcal{L}_1 distance, peak signal-to-noise ratio (PSNR), structural similarity index (SSIM) [36], and a learned perceptual metric LPIPS [43]. Following standard practice [27, 2], we compute PSNR and SSIM on image luma, and compute \mathcal{L}_1 distance and LPIPS directly on RGB values.

Baselines. We compare our model against both learning-based GatedConv [42] and three non-learning-based algorithms: PatchMatch [4], Image Melding [8], and Huang et al. [16]. All three non-learning algorithms are patch-based. Image Melding allows additional geometric and photometric transformations on patches. Huang et al. [16] first segments the image into different planes and augments a standard PatchMatch procedure with the plane rectification results.

Results. The results are summarized in Table 3 and Fig. 4. Here we run linear mixed models between each pair of methods and, for each metric, we mark in bold all methods that

| | | |
|---------------|---------------|---|
| Program | \rightarrow | CameraProgram; WorldProgram; |
| CameraProgram | \rightarrow | SetCamera(pos=Vec3, point_to=Vec3); |
| WorldProgram | \rightarrow | PlaneProgram; PlaneProgram; World Program; |
| PlaneProgram | \rightarrow | SetPlane(pos=Vec3, normal=Vec3) For1Stmt; |
| For1Stmt | \rightarrow | For (i in range(Integer, Integer)) { DrawStmt; For2Stmt } |
| For2Stmt | \rightarrow | For (j in range(Integer, Integer)) { DrawStmt } |
| DrawStmt | \rightarrow | Draw (x=Expr, y=Expr) |
| Expr | \rightarrow | Real \times i + Real \times j Real \times i |

Table 1: The domain-specific language (DSL) of box programs. Language tokens `For`, `If`, `Integer`, `Real`, and arithmetic/logical operators follow the Python convention. `Vec3` denotes 3D real vectors.

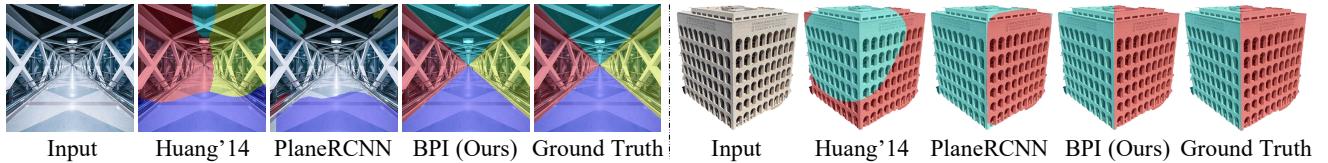


Figure 3: Visualization of the plane segmentation by different methods.

| Method | CrdO | CrdC | BldO | BldC |
|-------------------|-------------|-------------|-------------|-------------|
| Huang et al. [16] | 0.30 | 0.30 | 0.73 | 0.69 |
| PlaneRCNN | 0.52 | 0.52 | 0.63 | 0.63 |
| BPI (Ours) | 0.84 | 0.84 | 0.86 | 0.86 |

Table 2: Plane segmentation results in IoU between detected and groundtruth planes. BPI outperforms both baselines on both original (CrdO, BldO) and corrupted images (CrdC, BldC).

are indistinguishable with the best one. All p -values are in the supplementary material. Our method outperforms baselines on corridor images and achieves comparable results on building images with Huang et al. [16]. As discussed in Sec. F.2, Huang et al. [16] relies on straight lines on the plane to segment and rectify the image, so it works well on planes with a plethora of such features. Huang et al. [16] tends to fail on images without dense straight lines on the plane (rows 3-4 of Fig. 4). On corridor images, beyond producing high-fidelity inpainting results, our regularity-aware PatchMatch process preserves the structure of the scene, such as the light on the ceiling in row 1 of Fig. 4.

We also provide more qualitative results for image inpainting in Fig. 8 and image extrapolation in Fig. 9.

F.4. Time complexity.

For the corridor dataset, statistically, each image contains 1,506 wireframe combinations on average. 46 programs are evaluated on each plane. Note that BPI reduces the search space significantly based on the box prior, so that the search can be done efficiently ($23\times$ faster than without the box prior).

We also show the runtime of different algorithms on the task of plane segmentation (Table 4) and image inpainting (Table 5). The Image Melding [8] and Huang et al. [16] baselines are tested on a single machine with an Intel i7-6500U@2.5GHz CPU and 8GB RAM. All other baselines are tested on a single machine with an Intel E5-2650@2.20GHz CPU, a GeForce GTX 1080 GPU, and 8GB RAM.

F.5. Failure Case

Fig. 5 shows three main failure cases of our model. First, our model might misdetect vanishing points and wireframe segments, as illustrated in (a), where the model missed the wireframe between the floor and the right plane (detected wireframes shown as red lines). A second type of failure can occur when the image has a solid color plane. Illustrated in (b), our model segments the pure white part of the floor as part of the left/right planes. Finally, the inference might fail due to irregular planes, as shown in (c), where the buildings on the left do not form a rectangular plane. These issues could be mitigated with light user interaction, such as specifying wireframes.

References

- [1] Michael Ashikhmin. Synthesizing Natural Textures. In *I3D*, 2001. 1
- [2] Johannes Ballé, Valero Laparra, and Eero Simoncelli. End-to-end Optimized Image Compression. In *ICLR*, 2017. 4
- [3] Coloma Ballester, Marcelo Bertalmio, Vicent Caselles, Guillermo Sapiro, and Joan Verdera. Filling-in by Joint Interpolation of Vector Fields and Gray Levels. *IEEE TIP*, 10(8):1200–1211, 2001. 1

| Method | Corridors | | | | Buildings | | | | | |
|-------------------|-----------------|--------------|---------------|---------------|--------------|-----------------|---------------|---------------|--------|---------|
| | \mathcal{L}_1 | Mean ↓ | PSNR ↑ | SSIM ↑ | LPIPS ↓ | \mathcal{L}_1 | Mean ↓ | PSNR ↑ | SSIM ↑ | LPIPS ↓ |
| PatchMatch | 53.12 | 31.55 | 0.9872 | 0.0215 | 34.88 | 32.99 | 0.9896 | 0.0072 | | |
| Image Melding | 58.50 | 30.54 | 0.9880 | 0.0174 | 49.10 | 30.42 | 0.9890 | 0.0134 | | |
| Huang et al. [16] | 51.88 | 31.41 | 0.9869 | 0.0177 | 26.10 | 34.87 | 0.9917 | 0.0049 | | |
| GatedConv | 44.98 | 32.32 | 0.9883 | 0.0153 | 55.31 | 29.54 | 0.9866 | 0.0112 | | |
| BPI (Ours) | 38.45 | 34.21 | 0.9892 | 0.0170 | 26.43 | 34.86 | 0.9913 | 0.0054 | | |

Table 3: We compare BPI-guided PatchMatch with both patch-based and learning-based methods on the task of image inpainting. ↑ indicates that the higher the number, the better. **Bold** indicates models that are indistinguishable with the best one under that metric with a linear mixed model. See text for details.

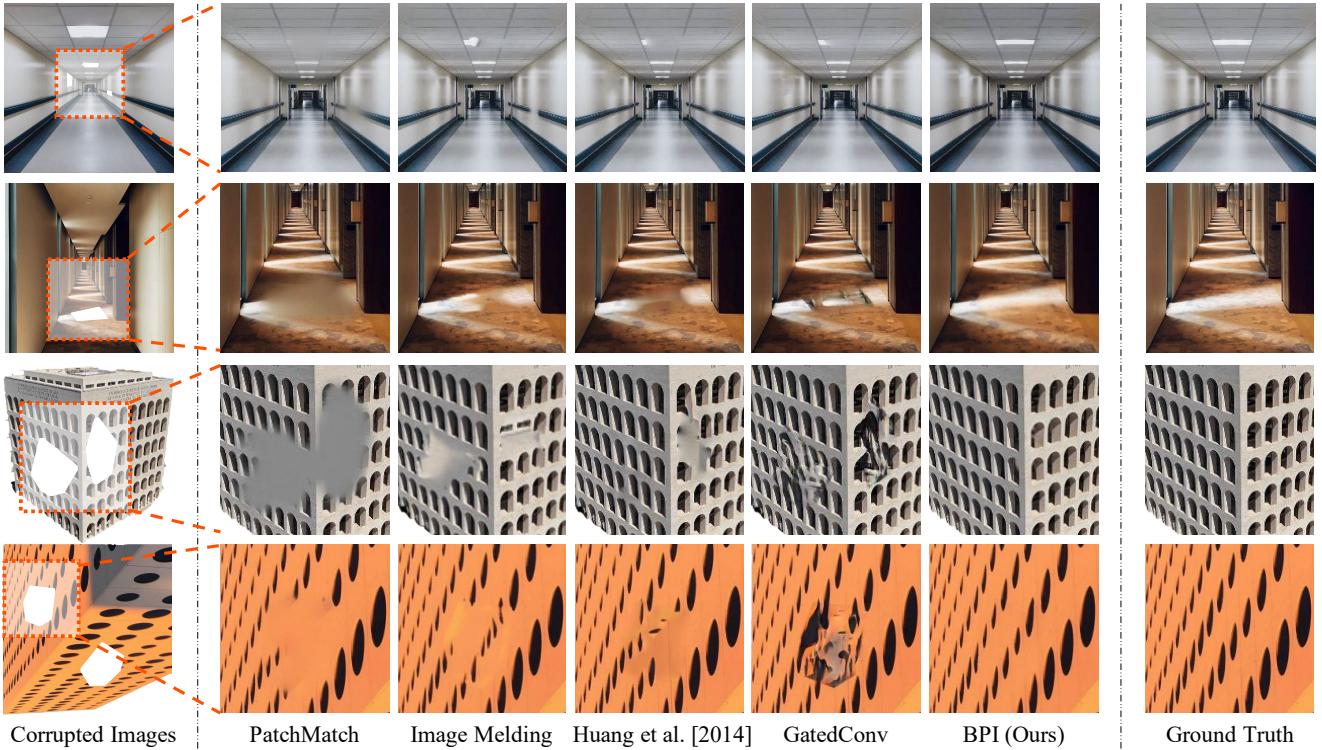


Figure 4: Qualitative results on the task of image inpainting. Compared with the baselines, our model can better preserve the regular structures even if they are sparse or have strong perspective effects.

- [4] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. *ACM TOG*, 28(3):24, 2009. [1](#), [4](#), [7](#)
- [5] Tony Beltramelli. Pix2Code: Generating Code from a Graphical User Interface Screenshot. In *EICS*, 2018. [1](#)
- [6] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *CVPR*, 2017. [4](#)
- [7] Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B. Goldman, and Pradeep Sen. Image melding: combining inconsistent images using patch-based synthesis. *ACM TOG*, 31:82:1–82:10, 2012. [1](#)
- [8] Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B. Goldman, and Pradeep Sen. Image Melding: Combining Inconsistent Images using Patch-based Synthesis. In *SIGGRAPH*, 2012. [4](#), [5](#), [7](#)
- [9] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. Image-to-Markup Generation with Coarse-to-Fine Attention. In *ICML*, 2017. [1](#)
- [10] Alexei A. Efros and William T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *SIGGRAPH*, 2001. [1](#)
- [11] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to Infer Graphics Programs from Hand-Drawn Images. In *NeurIPS*, 2018. [1](#)
- [12] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Esfahlami, and Oriol Vinyals. Synthesizing Programs for Images Using Reinforced Adversarial Learning. In *ICML*, 2018. [1](#)

| | Huang et al. [16] | PlaneRCNN [22] | BPI (Ours) |
|----------------|-------------------|----------------|------------|
| Corridor Boxes | 1.91s | 0.14s | 43.50s |
| Building Boxes | 6.83s | 0.76s | 171.20s |

Table 4: Runtime of different methods on the task of plane segmentation.

| | PatchMatch [4] | Image Melding [8] | Huang et al. [16] | Gated Conv [42] | BPI (Ours) |
|----------------|----------------|-------------------|-------------------|-----------------|------------|
| Corridor Boxes | 61.6s | 1275.1s | 18.9s | 1.9s | 20.1s |
| Building Boxes | 122.1s | 808.3s | 64.4s | 4.9s | 41.4s |

Table 5: Runtime of different methods on the task of image inpainting.

- [13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. [4](#)
- [14] Kaiming He and Jian Sun. Statistics of Patch Offsets for Image Completion. In *ECCV*, 2012. [1](#)
- [15] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering the Spatial Layout of Cluttered Rooms. In *CVPR*, 2009. [2](#)
- [16] Jia-Bin Huang, Sing Bing Kang, Narendra Ahuja, and Johannes Kopf. Image Completion using Planar Structure Guidance. *ACM TOG*, 33:129:1–129:10, 2014. [1, 4, 5, 6, 7](#)
- [17] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and Locally Consistent Image Completion. *ACM TOG*, 36(4):107, 2017. [1](#)
- [18] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. GRASS: Generative Recursive Autoencoders for Shape Structures. *ACM TOG*, 36(4):52, 2017. [1](#)
- [19] Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. GRAINS: Generative Recursive Autoencoders for INdoor Scenes. *ACM TOG*, 38(2):12:1–12:16, 2019. [1](#)
- [20] Yikai Li, Jiayuan Mao, Xiuming Zhang, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Perspective Plane Program Induction from a Single Image. In *CVPR*, 2020. [1, 4](#)
- [21] David Liebowitz, Antonio Criminisi, and Andrew Zisserman. Creating Architectural Models from Images. *CGF*, 18(3):39–50, 1999. [2](#)
- [22] Chen Liu, Kihwan Kim, Jinwei Gu, Yasutaka Furukawa, and Jan Kautz. PlaneRCNN: 3D Plane Detection and Reconstruction From a Single Image. In *CVPR*, 2019. [4, 7](#)
- [23] Guilin Liu, Fitzsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image Inpainting for Irregular Holes Using Partial Convolutions. In *ECCV*, 2018. [1](#)
- [24] Yunchao Liu, Zheng Wu, Daniel Ritchie, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to Describe Scenes with Programs. In *ICLR*, 2019. [1](#)
- [25] Jiayuan Mao, Xiuming Zhang, Yikai Li, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Program-Guided Image Manipulators. In *ICCV*, 2019. [1](#)
- [26] John FJ Mellor, Eunbyung Park, Yaroslav Ganin, Igor Babuschkin, Tejas Kulkarni, Dan Rosenbaum, Andy Ballard, Theophane Weber, Oriol Vinyals, and SM Eslami. Unsupervised Doodling and Painting with Improved SPIRAL. *arXiv:1910.01007*, 2019. [1](#)
- [27] Philipp Merkle, Aljoscha Smolic, Karsten Muller, and Thomas Wiegand. Multi-view Video Plus Depth Representation and Coding. In *ICIP*, 2007. [4](#)
- [28] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Qureshi, and Mehran Ebrahimi. EdgeConnect: Generative Image Inpainting with Adversarial Edge Learning. *arXiv:1901.00212*, 2019. [1](#)
- [29] Chengjie Niu, Jun Li, and Kai Xu. Im2Struct: Recovering 3d Shape Structure from a Single Rgb Image. In *CVPR*, 2018. [1](#)
- [30] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. SinGAN: Learning a Generative Model from a Single Natural Image. In *ICCV*, 2019. [1](#)
- [31] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: Neural Shape Parser for Constructive Solid Geometry. In *CVPR*, 2018. [1](#)
- [32] Assaf Shocher, Shai Bagon, Phillip Isola, and Michal Irani. InGAN: Capturing and Remapping the “DNA” of a Natural Image. In *ICCV*, 2019. [1](#)
- [33] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to Infer and Execute 3D Shape Programs. In *ICLR*, 2019. [1](#)
- [34] A. Vedaldi and B. Fulkerson. VLFeat: An Open and Portable Library of Computer Vision Algorithms. <http://www.vlfeat.org/>, 2008. [4](#)
- [35] Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiquan Cheng, and Yueshan Xiong. Symmetry Hierarchy of Man-Made Objects. *CGF*, 30(2), 2011. [1](#)
- [36] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE TIP*, 13(4):600–612, 2004. [4](#)
- [37] Wei Xiong, Zhe Lin, Jimei Yang, Xin Lu, Connally Barnes, and Jiebo Luo. Foreground-Aware Image Inpainting. In *CVPR*, 2019. [1](#)
- [38] Zhaoqi Yan, Xiaoming Li, Mu Li, Wangmeng Zuo, and Shiguang Shan. Shift-Net: Image Inpainting Via Deep Feature Rearrangement. In *ECCV*, 2018. [1](#)

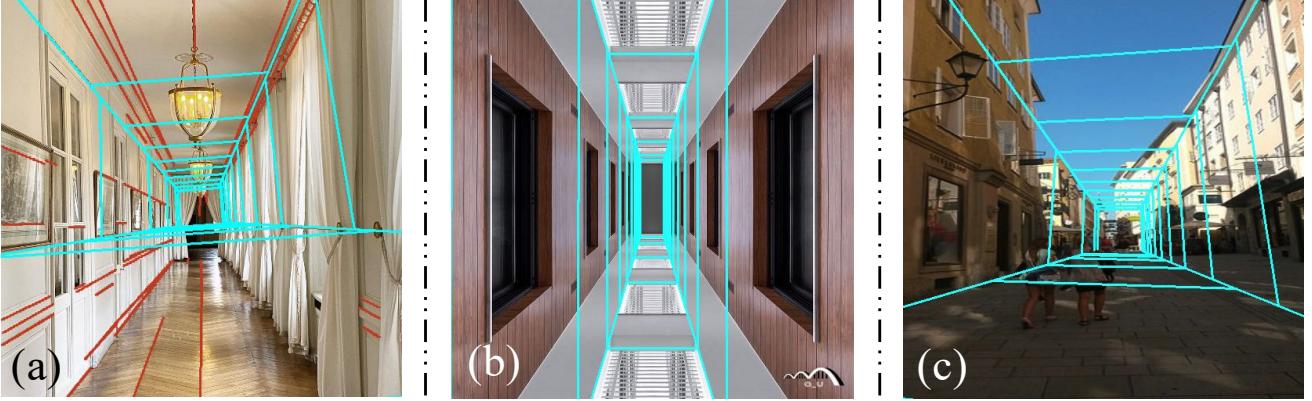
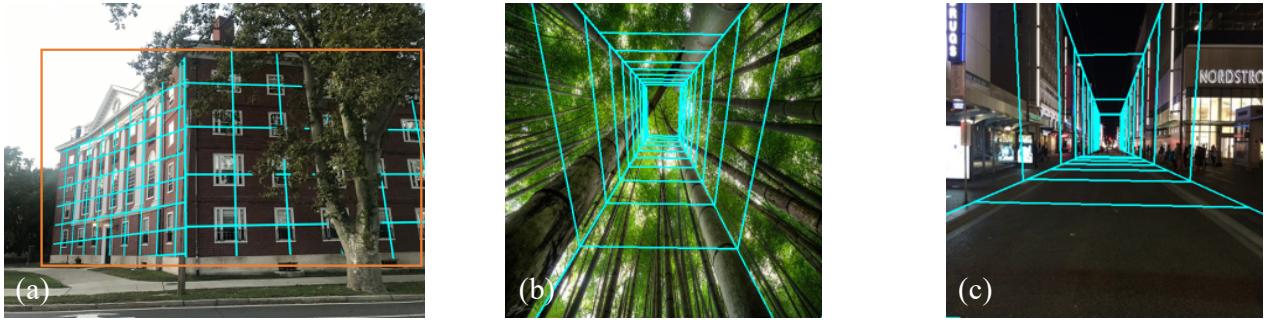


Figure 5: Failure cases. Our model may fail when neural networks misdetect visual cues (a). When image contains a solid color plane, our model may segment the pure white part to other planes (b). The inference might fail on irregular scenes, but could be mitigated with user interaction (c).



```

SetPlane_1(pos=[-1.4, -1.48, 1.68],
           normal=[0.40, 0.06, 0.92])
for i in range(0, 6)
    for j in range(0, 9)
        Draw( [ ] , x = -28 * i
              + 6 * j,
              y = 52 * j)
SetPlane_r(pos=[-1.5, -1.49, 1.10],
           normal=[-0.07, 0.14, 0.99])
for i in range(0, 5)
    for j in range(0, 5)
        Draw( [ ] , x = 55 * i,
              y = 52 * j)

```

```

SetPlane_1(pos=[-1.1, -1.39, 0.23],
           normal=[-1.00, -0.01, -0.00])
for i in range(0, 10)
    Draw( [ ] , x = 20 * i)
SetPlane_t(pos=[1.04, -1.37, 0.23],
           normal=[0.01, -0.95, -0.31])
for i in range(0, 10)
    Draw( [ ] , x = 20 * i)
SetPlane_r(pos=[1.00, 1.55, 1.19],
           normal=[1.00, 0.01, 0.00])
for i in range(0, 10)
    Draw( [ ] , x = 20 * i)
SetPlane_b(pos=[-1.15, 1.52, 1.19],
           normal=[-0.01, 0.95, 0.31])
for i in range(0, 10)
    Draw( [ ] , x = 20 * i)

```

```

SetPlane_1(pos=[-1.0, -2.77, 0.10],
           normal=[-1.00, -0.01, 0.03])
for i in range(0, 10)
    Draw( [ ] , x = 20 * i)
SetPlane_t(pos=[1.03, -2.75, 0.03],
           normal=[0.01, -0.99, -0.10])
for i in range(0, 3)
    Draw( [ ] , x = 65 * i)
SetPlane_r(pos=[1.01, 0.42, 0.36],
           normal=[1.00, 0.01, -0.03])
for i in range(0, 9)
    Draw( [ ] , x = 22 * i)
SetPlane_b(pos=[-1.02, 0.40, 0.42],
           normal=[-0.01, 0.99, 0.10])
for i in range(0, 6)
    Draw( [ ] , x = 30 * i)

```

Figure 6: More image examples and the corresponding programs. We use cyan lines to visualize the lattice structure on each plane.

- [39] Chao Yang, Xin Lu, Zhe Lin, Eli Shechtman, Oliver Wang, and Hao Li. High-Resolution Image Inpainting Using Multi-Scale Neural Patch Synthesis. In *CVPR*, 2017. 1
- [40] Halley Young, Osbert Bastani, and Mayur Naik. Learning Neurosymbolic Generative Models via Program Synthesis. In *ICML*, 2019. 1
- [41] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative Image Inpainting with Contextual Attention. In *CVPR*, 2018. 1
- [42] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-Form Image Inpainting with Gated Convolution. In *ICCV*, 2019. 1, 4, 7
- [43] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Networks As a Perceptual Metric. In *CVPR*, 2018. 4
- [44] Yichao Zhou, Haozhi Qi, Jingwei Huang, and Yi Ma.

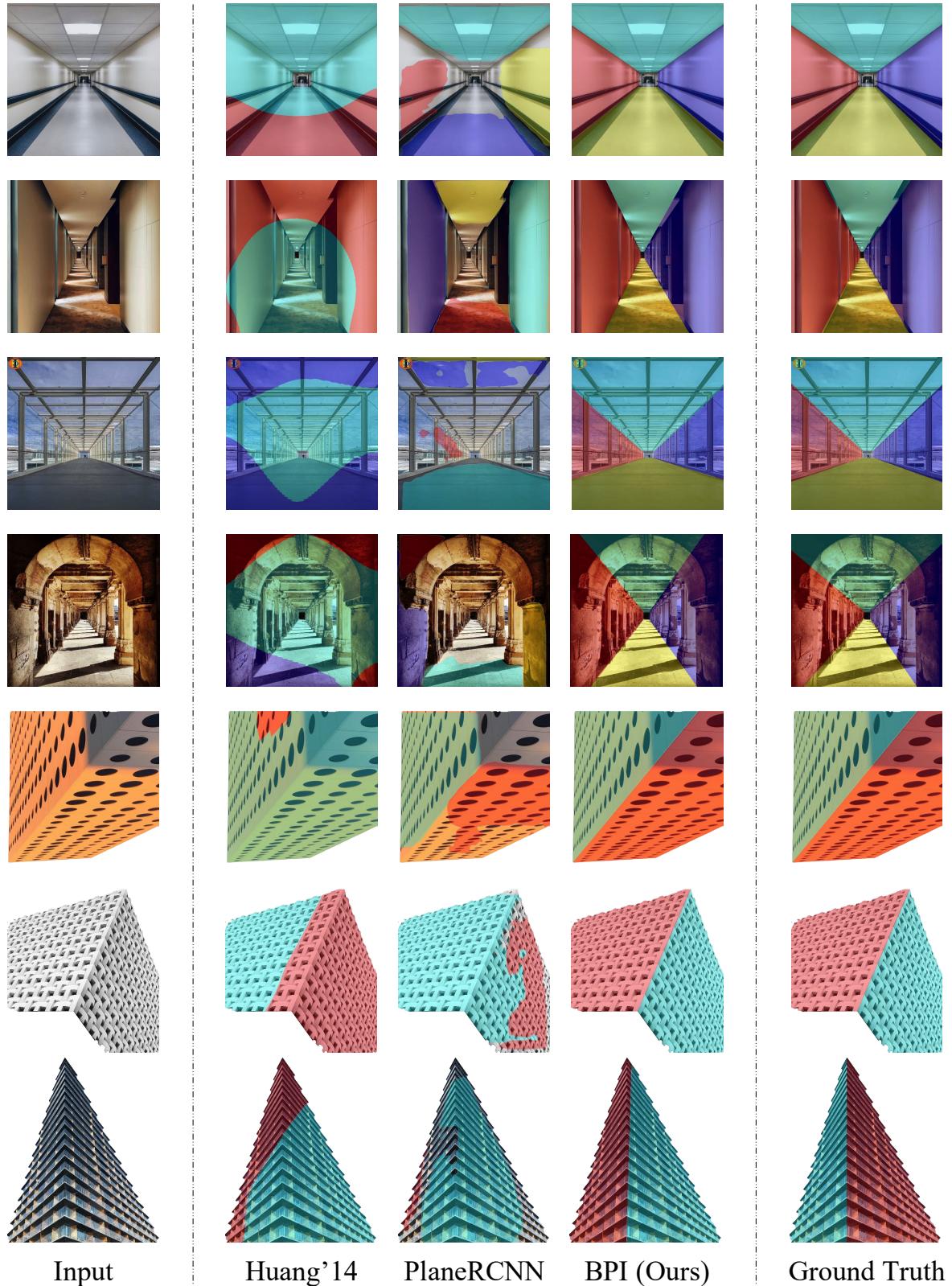


Figure 7: Visualization of the plane segmentation by different methods.

NeurVPS: Neural Vanishing Point Scanning via Conic Convolution. In *NeurIPS*, 2019. [2](#)

[45] Yichao Zhou, Haozhi Qi, and Yi Ma. End-to-end Wireframe Parsing. In *ICCV*, 2019. [2](#), [3](#)

[46] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-Stationary Texture Synthesis by Adversarial Expansion. *ACM TOG*, 37(4):49, 2018. [1](#)

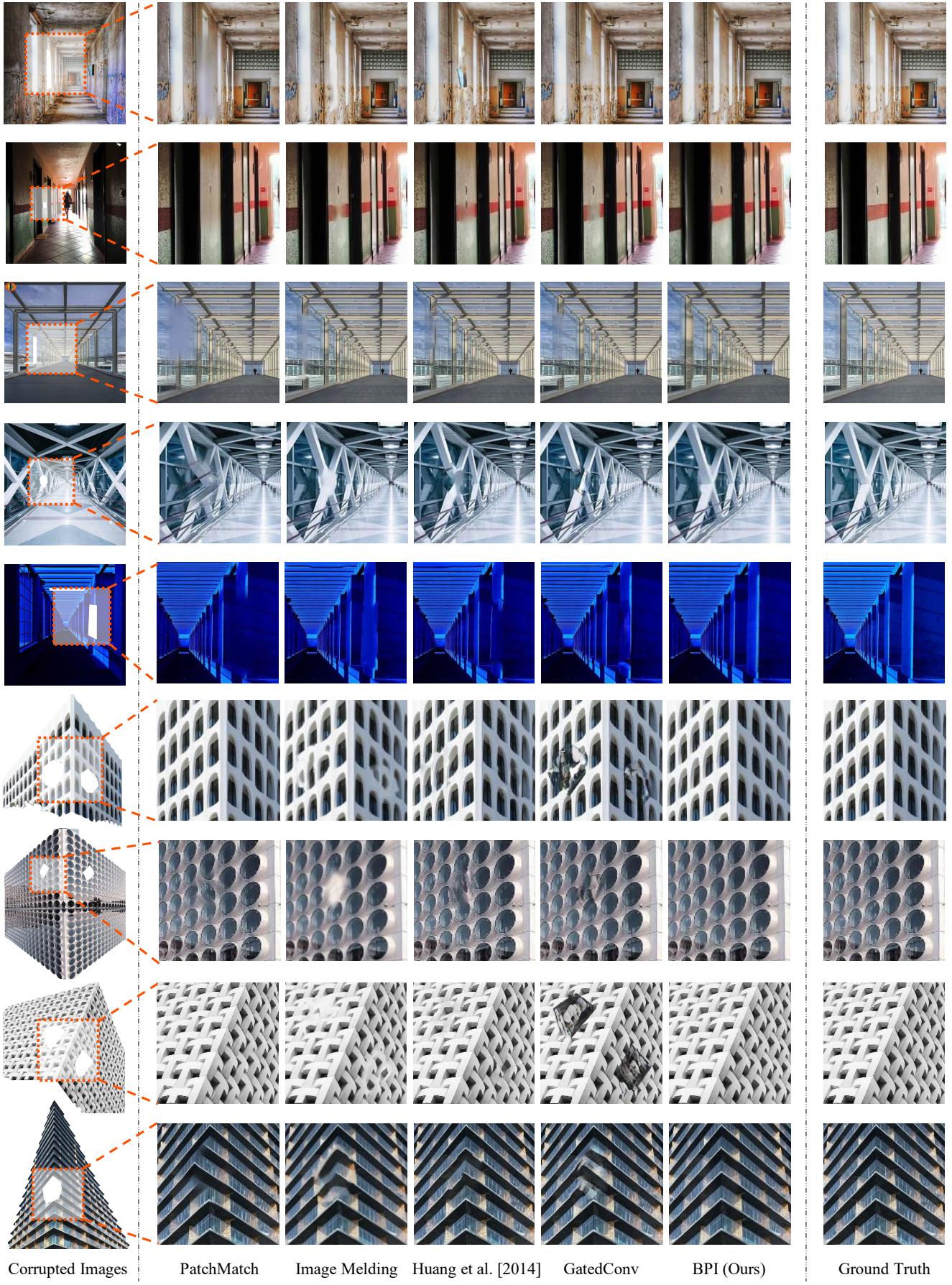


Figure 8: Qualitative results on the task of image inpainting.



Figure 9: Qualitative results on the task of image extrapolation.