Multi-Plane Program Induction with 3D Box Priors

Yikai Li[1,2*] Jiayuan Mao[1*] Xiuming Zhang[1]
William T. Freeman[1,3] Joshua B. Tenenbaum[1] Noah Snavely[3] Jiajun Wu[4]
[1]MIT CSAIL [2]Shanghai Jiao Tong University [3]Google Research [4]Stanford University
http://bpi.csail.mit.edu

Abstract

We consider two important aspects in understanding and editing images: modeling regular, program-like texture or patterns in 2D planes, and 3D posing of these planes in the scene. Unlike prior work on image-based program synthesis, which assumes the image contains a single visible 2D plane, we present Box Program Induction (BPI), which infers a program-like scene representation that simultaneously models repeated structure on multiple 2D planes, the 3D position and orientation of the planes, and camera parameters, all from a single image. Our model assumes a *box prior*, i.e., that the image captures either an *inner view* or an *outer view* of a box in 3D. It uses neural networks to infer visual cues such as vanishing points or wireframe lines to guide a search-based algorithm to find the program that best explains the image. Such a holistic, structured scene representation enables 3D-aware interactive image editing operations such as inpainting missing pixels, changing camera parameters, and extrapolate the image contents.

1. Introduction

We aim to build autonomous algorithms that can infer two important structures for compositional scene understanding and editing from a single image: the regular, program-like texture or patterns in 2D planes and the 3D posing of these planes in the scene. As a motivating example, when observing a single image of a corridor like the one in Fig. 1, we humans can effortlessly infer the camera pose, partition the image into five planes—including left and right walls, floor, ceiling, and a far plane—and recognize the repeated pattern on each of these planes. Such a holistic and structural representation allows us to flexibly edit the image, for instance by inpainting missing regions, moving the camera, and extrapolating the corridor to make it infinite.

A range of computer vision algorithms have utilized such a holistic scene representation to guide image manipulation tasks. Several recent ones fit into a program-guided image manipulation framework [2, 6, 7]. These methods infer a program-like image representation that captures camera parameters and scene structures, enabling image editing operations guided by such programs so that the scene structure is preserved during editing. However, due to the combinatorial complexity of possible compositions of elementary components based on the program grammar, these methods usually only work for images in highly specific domains with a fixed set of primitives such as hand-drawn figures of simple 2D geometric shapes [2] and synthesized tabletop scenes [6], or natural images with of a *single* visible plane, such as ground tiles and patterned cloth [7, 5].

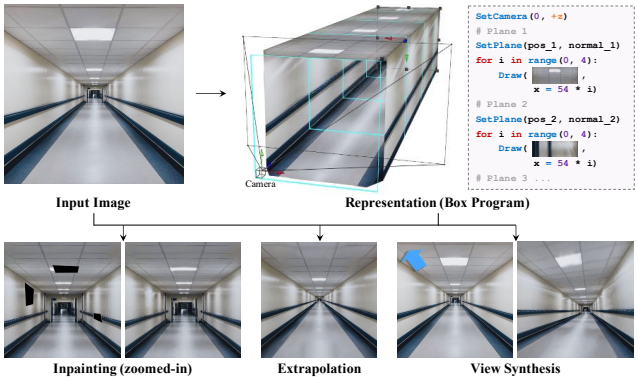
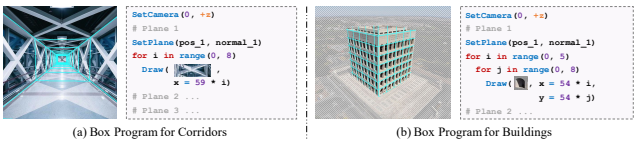


Figure 1: We present Box Program Induction (BPI), which infers a program-like scene representation that simultaneously models repeated structure on *multiple* 2D planes, 3D positions and orientations of the planes, relative to the camera, all from a *single* image. The inferred program can guide perspective- and regularity-aware image manipulation tasks: image inpainting, extrapolation, and view synthesis.

To address these issues and scale up program-guided image manipulation, we present a new framework, namely, Box Program Induction (BPI, for short), that jointly segments the image into multiple planes and infers the repeated structure on each plane. Our model assumes a *box prior*, leveraging the observation that box-like structures widely exist in images. Many indoor and outdoor scenes fall into this category: walking in a corridor or room corresponds to observing a box from the inside, and taking a picture of a building corresponds to seeing a box from the outside.

* indicates equal contribution.



(a) Box Program for Corridors (b) Box Program for Buildings

Figure 2: Example box programs inferred by BPI. They jointly model camera parameters, the 3D positions and orientations of multiple planes, as well as the regularity structure on planes.

To enable efficient inference of box programs, we also propose to utilize mid-level cues, such as vanishing points or wireframe lines, as well as high-level visual cues such as subject segmentations, to implicitly constrain the search space of candidate programs. Given the input image, BPI first infers these visual cues with pre-trained data-driven models. Next, it enumerates all candidate programs that satisfy the implicit constraints imposed by these inferred visual features. Finally, it ranks all candidate programs using low-level visual features such as pixel reconstruction.

In summary, we present BPI, a framework for inducing *box programs* from images by exploiting learned visual cues. Our experiments show that BPI can efficiently and accurately infer the structure and camera parameters for both indoor and outdoor scenes. The inference procedure is robust to errors and noise inherent to visual cue prediction: BPI automatically selects the best candidate wireframe lines and refines the vanishing points if they are not accurate. BPI also enables users to make 3D-aware interactive editing to images, such as inpainting missing pixels, extrapolating image content in specific directions, and changing camera parameters.

2. Box Program Induction

Our proposed framework, Box Program Induction (BPI), takes an image as input and infers a box program that best describes the image, guided by visual cues.

2.1. Domain-Specific Language

We use a domain-specific language (DSL), namely the *box programs*, to describe multiple planes in the scene and the regularity structure on individual planes. We assume these planes are faces of a 3D box. Shown in Fig. 1, a box program consists of two parts: camera parameters and programs for individual planes. A *plane program* first sets the plane's surface normal, then specifies a sub-program defining the regular 2D pattern on the plane. These patterns utilize the primitive `Draw` command, which places patterns at specified 2D positions. `Draw` commands appear in `For`-loop statements that characterize the regularity of each plane. We include the full DSL in the appendix.

2.2. Box Program Fitness

Given an input image, our goal is to segment the image into different planes, estimate their surface normals relative to the camera, and infer the regular patterns. We treat this

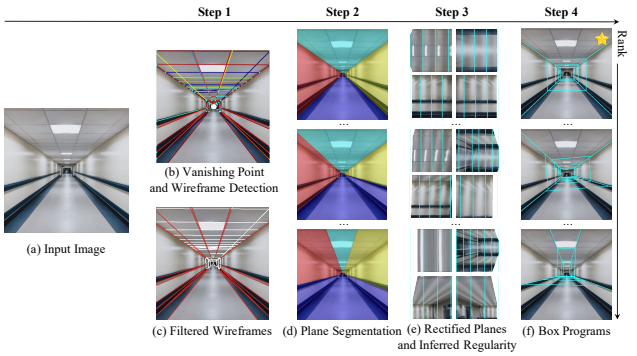


Figure 3: Our Box Program Induction finds the best-fit program that describes the input image (a). It first detects the vanishing point and wireframe segments (b), followed by a filtering step (c). It then constructs a set of candidate plane segmentation maps (d). Given each plane segmentation, it rectifies each plane and infers its regularity structure (e). We rank all candidate box programs by their fitness (f); the starred candidate is the best.

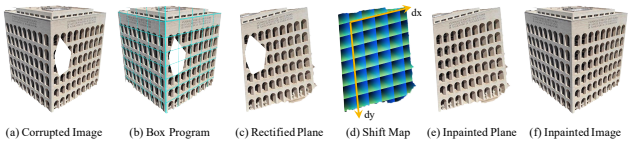
problem as seeking a program P that best fits the input image I. We first define the *fitness* of a program P by measuring how well P reconstructs I. Recall that a box program P is composed of multiple *plane programs*, each of which defines the regular pattern on a plane, its position, and orientation in 3D. Based on camera parameters, we can *rectify* each plane, resulting in images without perspective effects $\{J_1, J_2, \cdots, J_k\}$, where k is the number of planes in P.

For each plane i, we compute its fitness score by comparing its rectified image J_i and the corresponding plane program block Q_i. The score is defined similarly as in the Perspective Plane Program Induction framework [5]. Specifically, executing Q_i produces a set of 2D coordinates C_i that can be interpreted as the centers of each visual element. Since Q_i contains a nested loop of up to two levels, we denote the loop variable for each `For` loop as a and b. Thus, each 2D coordinate in C_i can be written as a function of a and b, $c(a, b) \in \mathbb{R}^2$. The fitness score F is defined as, $F = -\sum_{a,b} \big[\|J_i[c(a,b)] - J_i[c(a+1,b)]\|_2^2 + \|J_i[c(a,b)] - J_i[c(a,b+1)]\|_2^2 \big]$, where $\|\cdot\|_2$ is the $\mathcal{L}_2$ norm and $J_i[c(a,b)]$ denotes the patch centered at $c(a,b)$.

2.3. 3D Box Priors and the Role of Visual Cues

A naïve way to find the best-fit program P is to enumerate all possible plane segmentations of the input image I and rank all candidates by the fitness score. However, the complexity of this naïve method scales exponentially in the number of planes. Instead, we consider 1) the *box prior*, which constrains the plane segmentation of the image, and 2) visual cues that help to guide the search. Specifically, we impose the following *box prior*:

- For an *inner view* of a box, e.g., a corridor as in Fig. 2a, our box program models four planes: two side walls, the floor, and the ceiling. For images with a far plane, we will



(a) Corrupted Image (b) Box Program (c) Rectified Plane (d) Shift Map (e) Inpainted Plane (f) Inpainted Image

Figure 4: An illustration of the proposed program-guided PatchMatch. Given the corrupted image, we first detect its box program (b) and rectify each plane (c). We use the regular structure (d) on the plane to guide the PatchMatch to inpaint the corrupted plane (e). The color in (d) visualizes the relative position of each pixel to the center of its associated lattice cell.

segment the far plane but do not use programs to model it, as most far planes do not contain a regular structure.

- For an *outer view* of a box, e.g., a building as in Fig. 2b, our box program models the two side walls as two planes. We do not model the roof as, in most images, the roof is either nonvisible or very small in size.

Below, we show how to use *visual cues* to guide the search for box programs. We focus on the *inner view* case, and include details for the *outer view* case in the supplemental material. The full search process is illustrated in Fig. 3, and consists of four steps.

Given an input image (Fig. 3a), we apply NeurVPS [10] to detect the vanishing point and L-CNN [11] to extract wireframes in the image. We use the most confident prediction of NeurVPS as the vanishing point and filter out invalid wireframe segments that are too short or do not intersect with the vanishing point. Next, we generate a set of candidate plane segmentations based on possible combinations of the detected wireframe segments. Then, for each segmented plane, we seek the program that best describes the plane structure based on the fitness score. Finally, we sum up the fitness score for all planes in a 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.

2.4. Program-Guided Image Manipulation

The inferred box program enables 3D-aware interactive image manipulation. Lying at the core of these applications is a program-guided PatchMatch algorithm.

Our program-guided PatchMatch augments the PatchMatch algorithm [1] by enforcing the regularity on each plane. Consider the task of inpainting missing pixels on the wall of a building in Fig. 4. Taking a corrupted image as input (Fig. 4a), we infer the box program from the undamaged pixels (Fig. 4b). The inferred box program explains the plane segmentation and the regularity pattern on each plane. For each plane that contains missing pixels, we can first rectify the corrupted plane as shown in Fig. 4c. Since the plane program on this plane will partition the image with a lattice, we construct a "shift map", illustrated as Fig. 4d, which represents the relative position of each pixel to the center of the lattice cell that this pixel lies in, normalized to $[-0.5, 0.5]$.

In PatchMatch, the similarity $sim(p, q)$ between pixel p and pixel q is computed as a pixel-level similarity sim_{pixel} between two patches centered at p and q, with a constant patch size δ_{pm}. We add another term to this similarity function: $sim(p, q) \triangleq sim_{\text{pixel}} + sim_{\text{reg}} = sim_{\text{pixel}} - \lambda_{\text{reg}} \|\text{wraparound}(smap[p] - smap[q])\|_2^2$, where λ_{reg} is a hyperparameter that controls the weight of the regularity enforcement. abs is the absolute value function. The shift-map term measures whether two pixels p and q correspond to the same location on two (possibly different) repeating elements on the plane. We "wrap around" shift map distance by $\text{wraparound}(\mathbf{x}) \equiv \max(1 - \mathbf{x}, \mathbf{x})$, as the top-left corner and the bottom-right corner of a cell also matches. Thus, the PatchMatch algorithm will choose the pixel based on both pixel similarity and regularity similarity to fill in the missing pixels (Fig. 4e).

3. Experiments

For evaluation, we introduce two datasets, and then apply box programs on these datasets. Here we focus on two tasks that showcase the advantage of perspective- and regularity-aware image manipulation based on programs: image extrapolation and view synthesis. We include additional evaluation of box programs on plane segmentation and image inpainting in the appendix.

Dataset. We collect two datasets from web image search engines for our experiments, a 44-image *Corridor Boxes* dataset and a 42-image *Building Boxes* dataset. These correspond to the inner view and the outer view of boxes, respectively. For both datasets, we manually annotate the plane segmentations by specifying edges of the boxes. For corridor images, we also create a mask for the far plane. For building images, we supplement the subject segmentation (i.e., the building of interest) to the dataset annotation.

3.1. Image Extrapolation

The inferred box programs support 3D-aware and regularity-preserving image manipulation, such as extrapolating box structures. Here, on the *Building Boxes* dataset, we show that our model can make the building taller or wider. The input to the model is a foreground mask of the building and the target region to be filled with the extrapolated building. We compare our method with four baselines: Content-Aware Scaling in Adobe Photoshop, Kaspar et al. [4], Huang et al. [3] and InGAN [8]. For content-aware scaling, we first select the foreground mask and then scale it so that it fills the target region. For both Kaspar et al. [4] and InGAN, we extract the bounding box of the foreground building and use it as the input. The model generates a new image that is 1.5x larger. For Huang et al. [3], we cast the extrapolation problem as inpainting the target region.

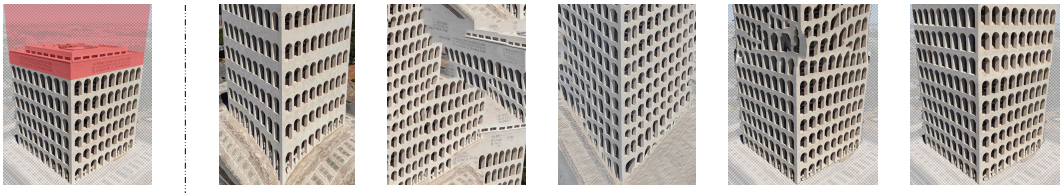


Figure 5: The "extrapolated" buildings. Content-Aware Scale, Kaspar et al. [4], and InGAN fail to preserve the building structure while extrapolating the image: they either generate irregular patterns or change the shape of the planes. Huang et al. [3] fails to preserve the regular structure when inpainting large areas.

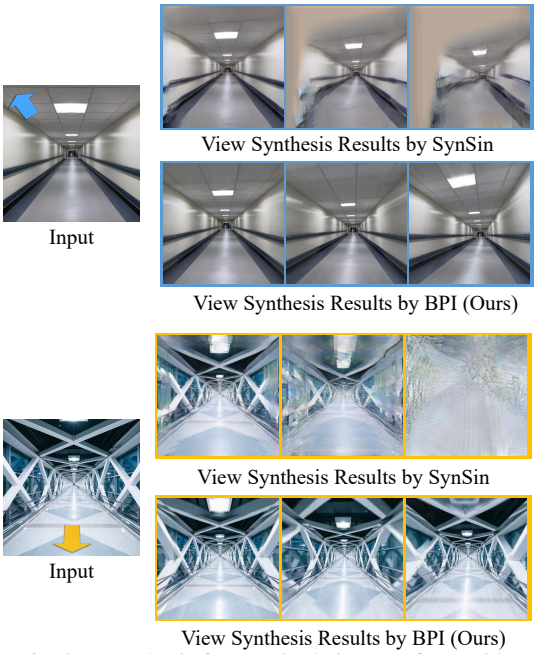


Figure 6: View synthesis from a single image of a corridor. Compared with the learning-based method SynSin, our model better preserves the regular patterns on the walls and also has remarkably fewer artifacts.

As shown in Fig. 5, Content-Aware Scaling is unaware of perspective effects and fails to preserve the lattice structure in the image. It also cannot generate new visual elements such as windows. Both Kaspar et al. [4] and InGAN do not preserve existing pixels when extrapolating the image. Kaspar et al. [4], as a texture synthesis method, also ignores the two-plane structure when generating the new image. While InGAN is able to capture the visual elements, it does not follow the lattice pattern during synthesis. Huang et al. [3] can inpaint small local areas as in Fig. **??**, but does not follow the regular structure when inpainting a large area. In contrast, BPI preserves both the plane structure of the building and the lattice pattern on each side.

Quantitatively, we randomly select 12 images, and ask 15 people to rank the outputs of different methods. We collected $12 \times 15 = 180$ responses. The preference for the models are: Ours(61%), Content-Aware Scaling(16%), Kaspar et al. [4] (2%), InGAN(16%), and Huang et al. [3] (5%).

3.2. View Synthesis

As our inferred box programs captures a holistic 3D scene representation, we can synthesize novel views of the scene from a *single* image. We compare different models on the *Corridor Boxes* dataset. We consider three types of camera movement: 1) "step into the corridor", 2) "step back in the corridor", and 3) "step back in the corridor, pan leftward, and tilt upward". All trajectories generate 5 frames. Detailed parameters are included in the supplemental material. Note that in both trajectories that involve "stepping back", the algorithm must synthesize pixels unseen in the original image. For our BPI, we run our program-guided PatchMatch to extrapolate the planes and synthesize pixels that are outside the input view frustum.

Results. We compare images generated by our BPI and by SynSin [9] in Fig. 6. The arrow on the input shows camera movement. As our method can generate a corridor of an arbitrary length, we see significantly fewer artifacts when the camera movement is large, compared with SynSin. Even in the top example where the camera movement is small, the pixels synthesized by SynSin that are outside the original view frustum already fail to preserve the regular structure.

For a quantitative comparison, we randomly select 20 images, generate synchronized videos of the results produced by our method and by SynSin on all three camera trajectories, and ask 10 people to rank the outputs. We collected $20 \times 3 \times 10 = 600$ responses. For the three different trajectories, 100%, 94%, and 99.5% of the responses prefer our result to that by SynSin, respectively.

4. Conclusion

We have presented Box Program Induction (BPI), a framework for inferring program-like representations that model the regular texture and patterns in 2D planes and the 3D posing of these planes, all from a single image. Our model assumes a *box prior*, which constrains the plane segmentation of the image, and uses visual cues to guide the inference. The inferred box program enables 3D-aware interactive image editing. Currently, our algorithm assumes that the full image can be partitioned into planes with regular structures. Future research may consider integrating models that can handle the presence of irregular image regions.

Acknowledgements

This work is supported by the Center for Brains, Minds and Machines (NSF STC award CCF-1231216), NSF #1447476, ONR MURI N00014-16-1-2007, and IBM Research. Work was done while Jiajun Wu was a visiting researcher at Google Research.

References

[1] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. *ACM TOG*, 28(3):24, 2009. 3

[2] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to Infer Graphics Programs from Hand-Drawn Images. In *NeurIPS*, 2018. 1

[3] 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. 3, 4

[4] Alexandre Kaspar, Boris Neubert, Dani Lischinski, Mark Pauly, and Johannes Kopf. Self Tuning Texture Optimization. *CGF*, 34(2):349–359, 2015. 3, 4

[5] 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, 2

[6] 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

[7] Jiayuan Mao, Xiuming Zhang, Yikai Li, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Program-Guided Image Manipulators. In *ICCV*, 2019. 1

[8] Assaf Shocher, Shai Bagon, Phillip Isola, and Michal Irani. InGAN: Capturing and Remapping the "DNA" of a Natural Image. In *ICCV*, 2019. 3

[9] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. SynSin: End-to-end View Synthesis from a Single Image. In *CVPR*, 2020. 4

[10] Yichao Zhou, Haozhi Qi, Jingwei Huang, and Yi Ma. NeurVPS: Neural Vanishing Point Scanning via Conic Convolution. In *NeurIPS*, 2019. 3

[11] Yichao Zhou, Haozhi Qi, and Yi Ma. End-to-end Wireframe Parsing. In *ICCV*, 2019. 3