Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks

Guy Katz, Clark Barrett, David Dill, Kyle Julian, Mykel Kochenderfer

Presented by: Atharva Sehgal

Slides available at: https://atharvas.net/static/reluplex

Online

- Overview of Neural Network Verification
  - Birds Eye Perspective
  - RELU Network formulation
  - RELU Network as linear constraints
- The Reluplex Algorithm
  - Simplex Algorithm
  - Reluplex as extension of Simplex
- Analysis
  - Reluplex and the model checking paradigm.
  - Issues

Neural Network Verification

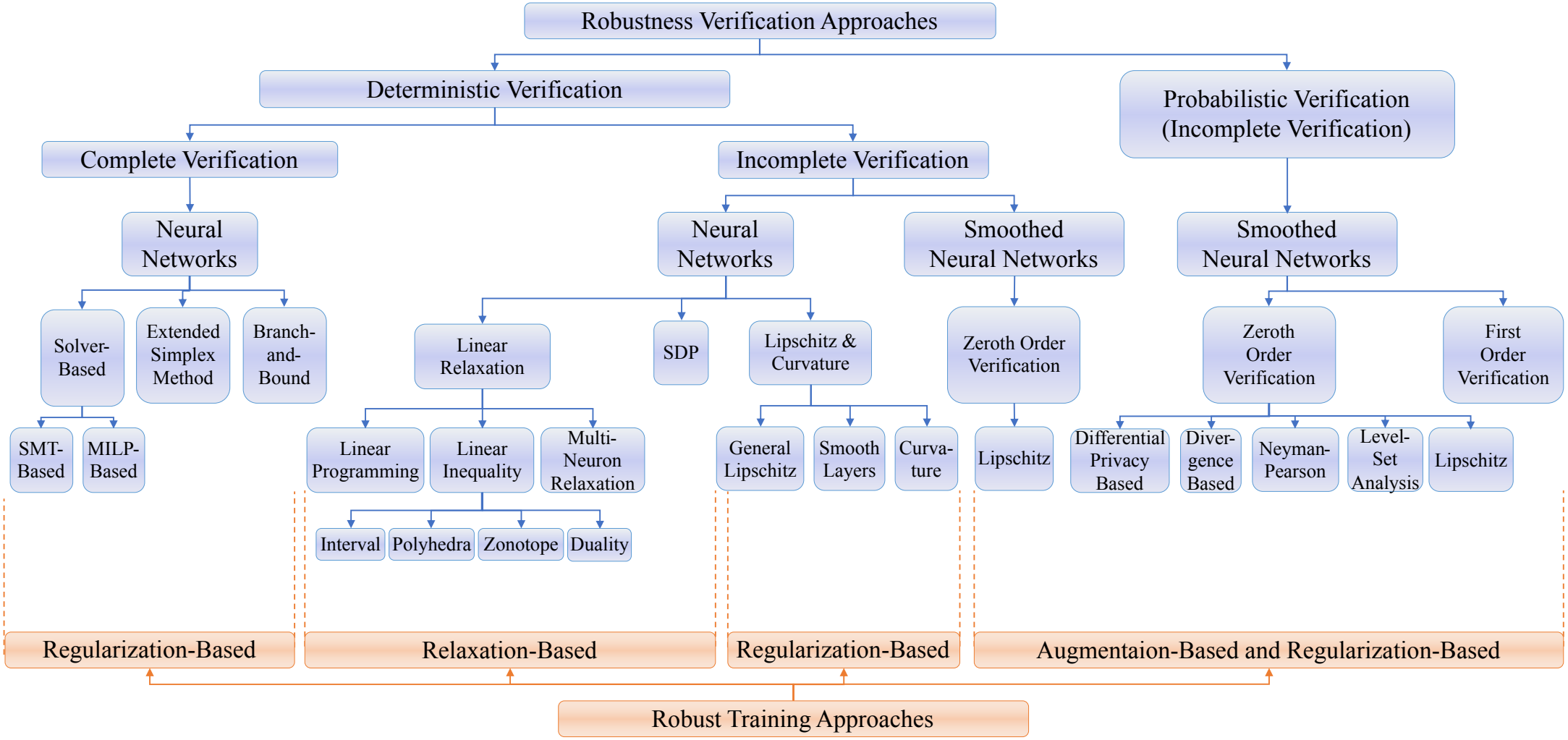
Given a NN f parametrized by θ, an input pair (X, y), a random labelling function $rand :: y$ We want to show that:

$$\min_x f(\theta, x)_y - f(\theta, x)_{rand(y)} > 0 \qquad \forall y \notin label(x)$$

- "We cannot find a violation to a “minimum margin” between the true prediction and any other prediction for a model."

usage: As surrogate functions. DNN replace lookup tables in aerospace engineering. SHOW DOD PICTURE.

Birds Eye perspective



https://sokcertifiedrobustness.github.io/

For datapoints $(x_0, y_0) \in (\mathcal{X} \times \mathcal{Y})$, neural network f_θ, feasible input region $B_p^\epsilon(x_0)$, and a threshold ϵ, A complete verification algorithm $\mathcal{A}(f_\theta, x_0, y_0, \epsilon)$ shows that:

$$\forall (x_0, y_0), \quad \exists x \in B_p^\epsilon(x_0) \quad \text{s.t} \quad f_\theta(x) \neq y_0$$
$$\implies \mathcal{A}(f_\theta, x_0, y_0, \epsilon) = \text{false}$$

ReLU Networks

For some input $x \in \mathcal{X}$, a feedforward neural network f parametrized by θ is defined by:

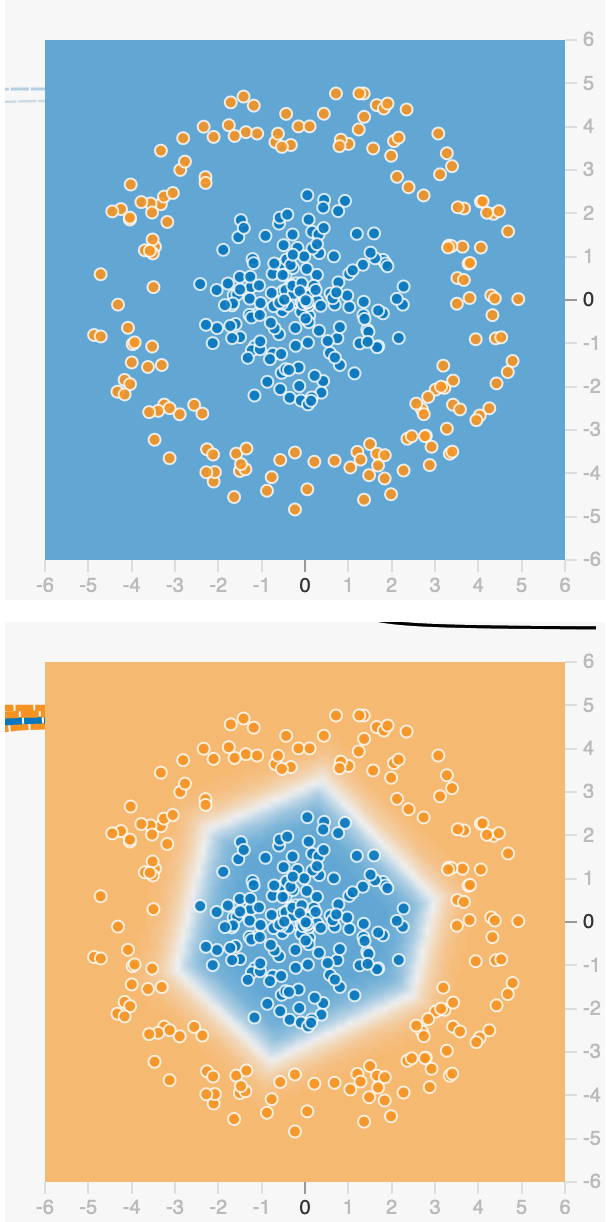
$$f_\theta(x) = \hat{z}_{(l)}$$

$$s.t. \begin{cases} z_{(1)} = x \\ \hat{z}_{(i+1)} = \mathbf{W}_i z_{(i)} + b_i & \forall i \in [1, \dots l-1] \\ z_{(i)} = \sigma(\hat{z}_{(i)}) & \forall i \in [2, \dots l] \end{cases}$$

A ReLU network is a specific instantiation of a feedforward network where the nonlinear function σ is $\mathrm{ReLU}(z)$ defined as:

$$\mathrm{ReLU}(x) = \max(0, x)$$

The output of a ReLU net is *locally linear* with respect to the input x.



Visualizations from playground.tensorflow.org

We will use the LRA theory for modelling a neural network. Specifically, in the case of a ReLU network:

- Each matrix multiplication can be encoded as a linear constraint:

$$l(z_{(l)}, \hat{z}_{(l)}) = \left(\hat{z}_{(l)} = \sum_{j=1}^{|z_{(l)}|} c_j \cdot z_{(l),j} + b \right)$$

- Each ReLU activation can be encoded as a disjunction:

$$\mathrm{ReLU}(\hat{z}_{(l)}, z_{(l)}) = \left(z_{(l)} > 0 \implies \hat{z}_{(l)} = z_{(l)}\right) \wedge \left(z_{(l)} \leq 0 \implies \hat{z}_{(l)} = 0\right)$$

$$\begin{aligned} x + y &\geq 4 \\ y &\geq 2 \end{aligned}$$

.

$$\begin{aligned} x + y &= s \\ y &\geq 2 \\ s &\geq 4 \end{aligned}$$

Thoery Solver: Linear Rational Arithmetic

Specifically, the simplex algorithm.

Intuition. Simplex working on a small example.

However, this only worked out well for us because the order was fixed.

Simplex Form

Example on the running example. Use slack variables

basic variable

non-basic variable

Convince yourself that, with this form, the ordering will always allow us to terminate.

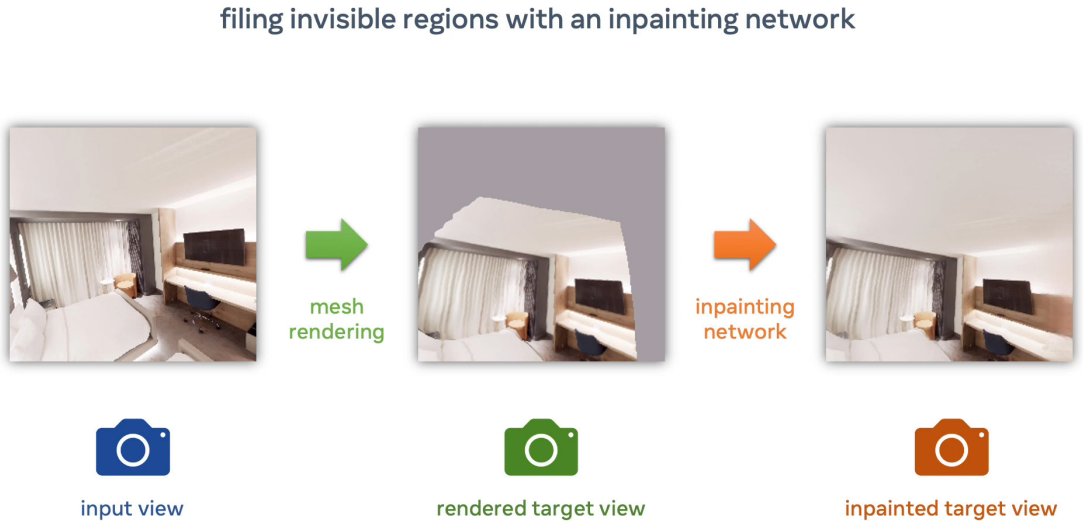
Reluplex algorithm

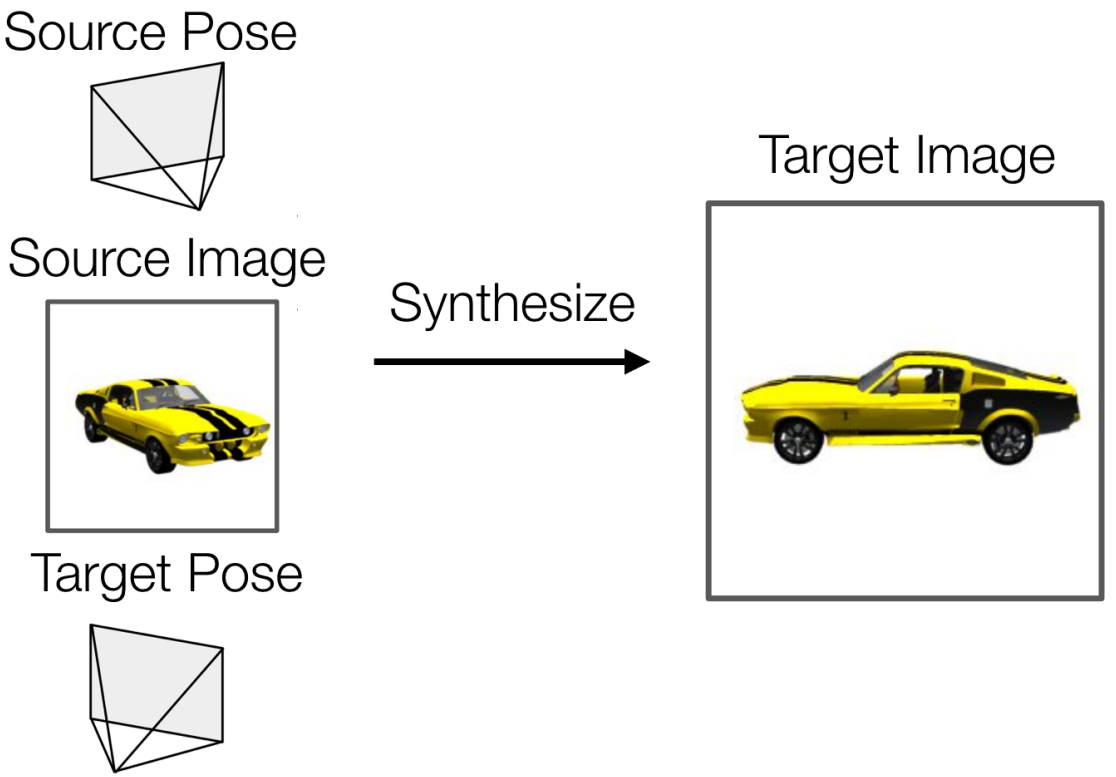
Naive Algorithm: Use Simplex as theory solver with a generic directly on relu conditions. Each relu condtion affects the relu dependencies later in the network. exponential blowup.

Intuition: We will only case split when necessary.

Motivation

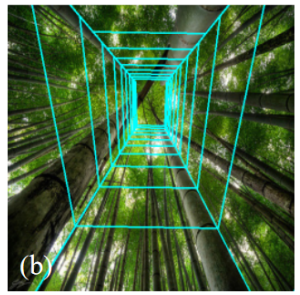
- Task: Single Image View Synthesis
- Challenge: Learning scene decompositions is, generally, underspecified.
- Observation: Physical objects display a measure of uniformity.
- Hypothesis: Can we impose a structural prior to capture this uniformity?
- Use a "box prior"
  - Pinhole camera
  - $\exists$ 4 planes if inside else 2 planes
  - Each plane has repeating pattern.



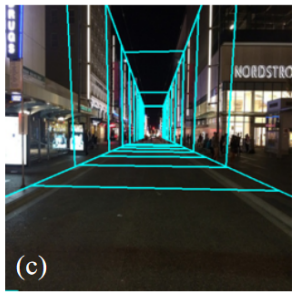


Sample Box Programs


```
SetPlane_l(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_l(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_l(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 11: More image examples and the corresponding programs. We use cyan lines to visualize the lattice structure on each plane.

Program	⟶ CameraProgram; WorldProgram;
CameraProgram	⟶ SetCamera(pos=`Vec3`, point_to=`Vec3`);
WorldProgram	⟶ PlaneProgram; | PlaneProgram; World Program;
PlaneProgram	⟶ SetPlane(pos=`Vec3`, normal=`Vec3`) For1Stmt;
For1Stmt	⟶ `For` (i `in range`(Integer, Integer)){ DrawStmt; | For2Stmt }
For2Stmt	⟶ `For` (j `in range`(Integer, Integer)){ DrawStmt }
DrawStmt	⟶ `Draw` (x=Expr, y=Expr)
Expr	⟶ `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.

Algorithm

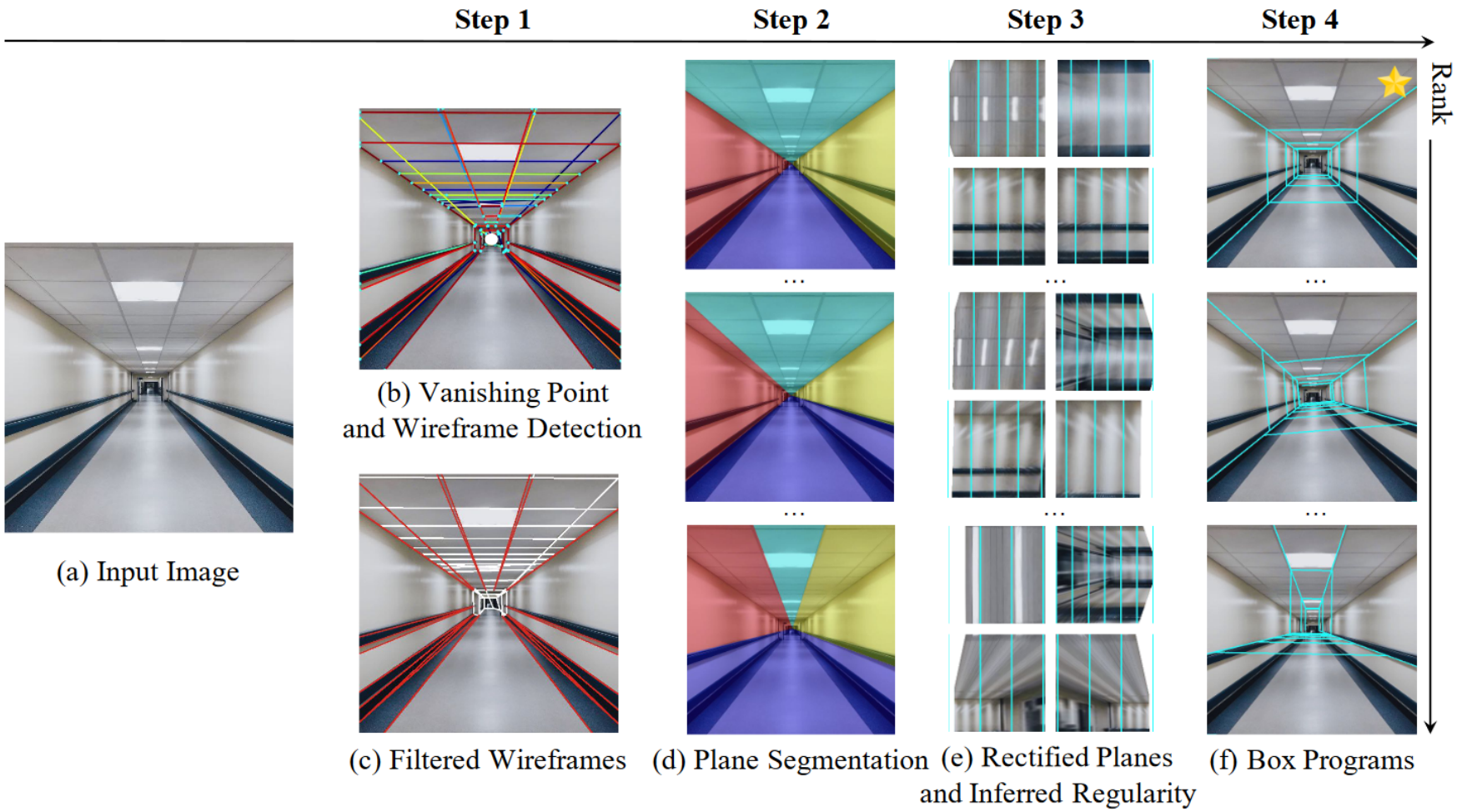
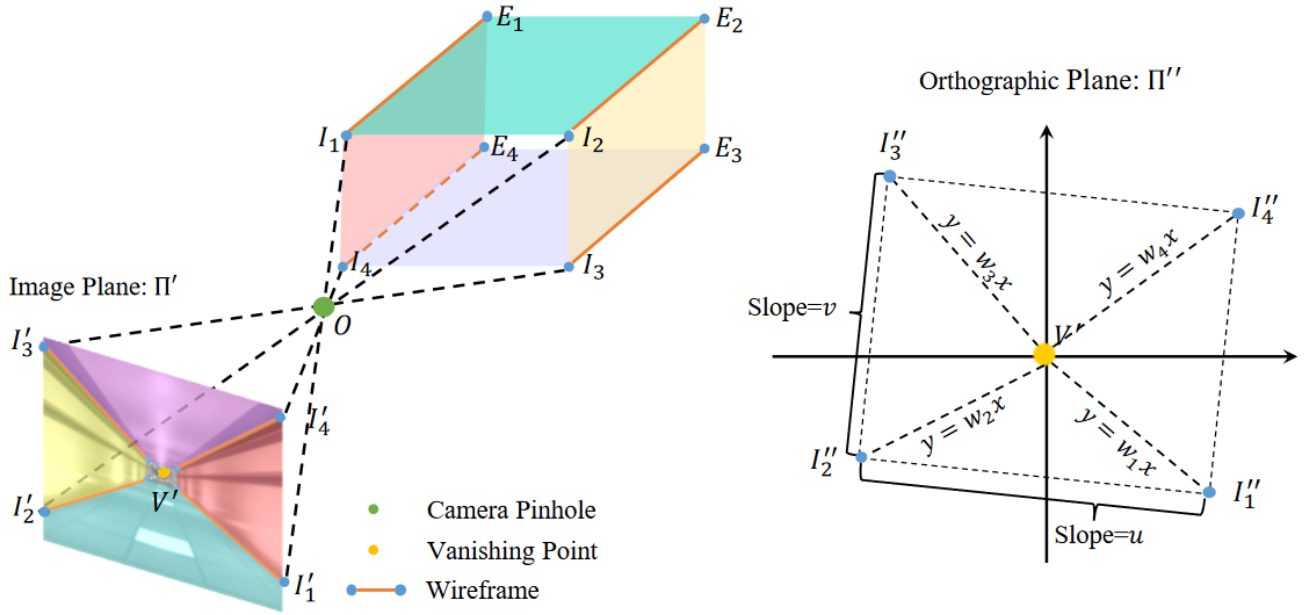


Figure 3: Our Box Program Induction finds the best-fit program that describes the input image (a). It first detects the vanishing point and wireframe segments (b), followed by a filtering step (c). It then constructs a set of candidate plane segmentation maps (d). Given each plane segmentation, it rectifies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

Step 3 - Plane Recticiation



(a) Perspective projection in an *inner view*

(b) Orthographic projection from the vanishing point

Figure 10: 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.

equation system:

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

Step 4 - Fitness Ranking

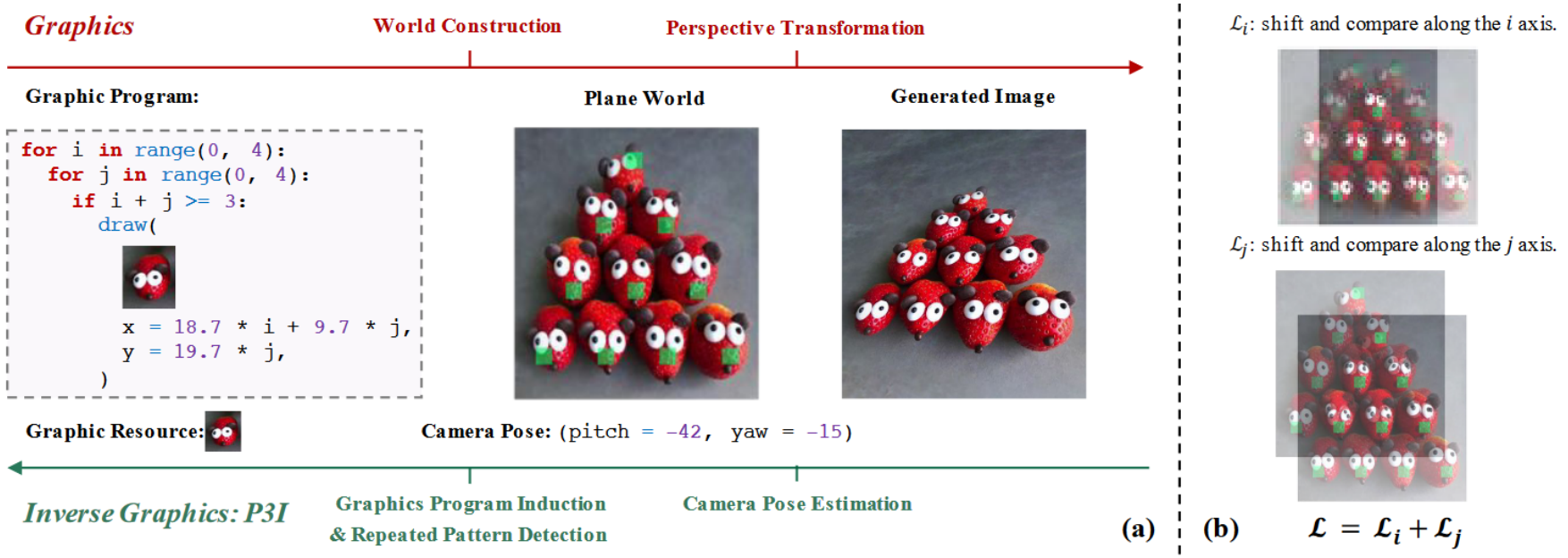


Figure 2: **(a)** Our model P3I solves an *inverse graphics* problem. Given an input image, P3I jointly infers the camera pose, object locations, and the global scene regularity, which is an inversion of a simplified graphics pipeline. **(b)** We compute the fitness of a program on the image based on a shift-and-compare routine, which we illustrates on a lattice pattern case.

Evaluation

Dataset

- 44 Corridor images, 42 Buildings.
- Collected by Google Images
- Handmade mask for end of the corridor
- Handmade mask for building.

Tasks

- Plane Segmentation
- Image Inpainting
- Image Extrapolation
- View Synthesis

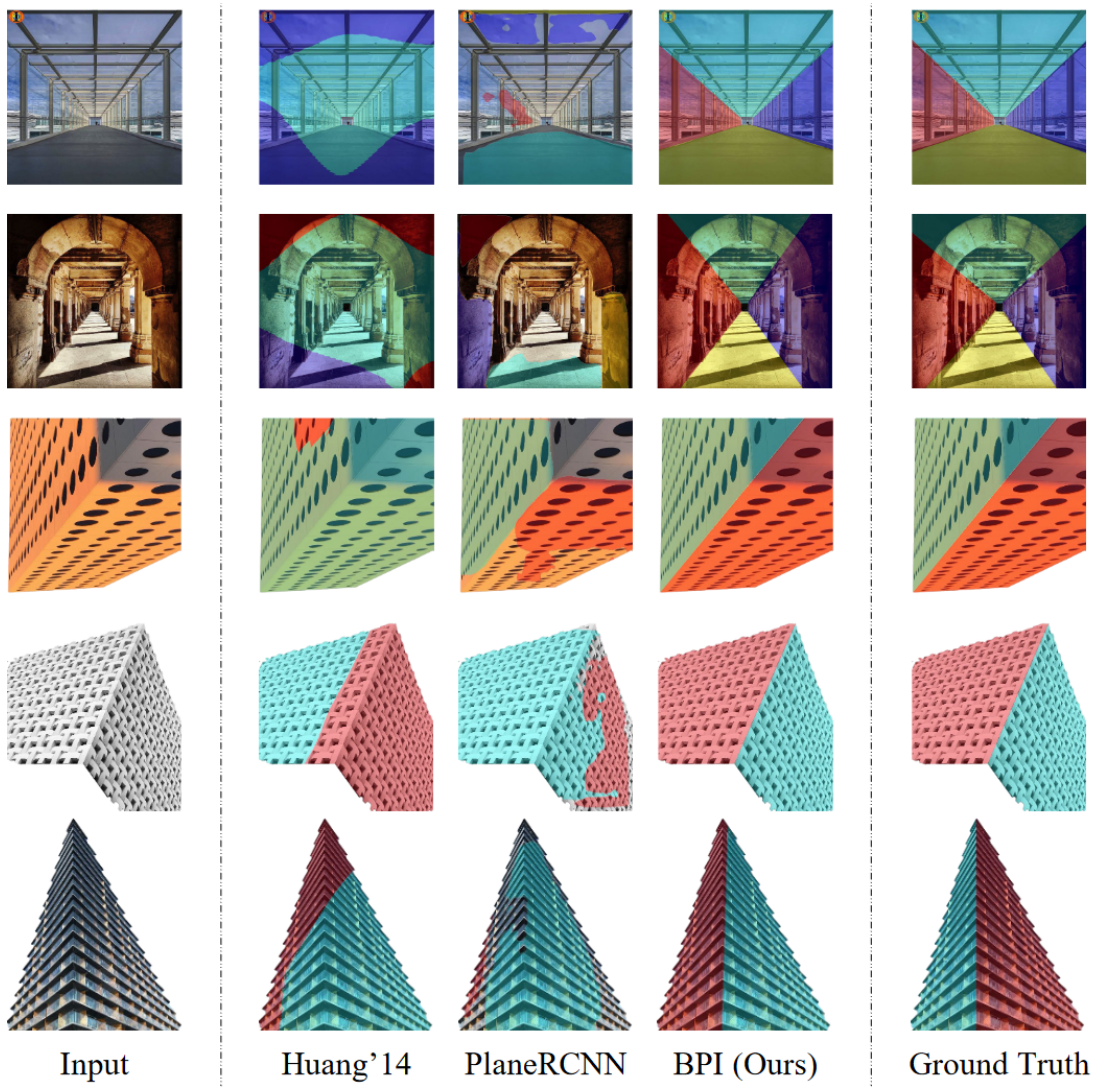


Figure 12: Visualization of the plane segmentation by different methods.

Plane Segmentation (Quantitiave Evaluation)

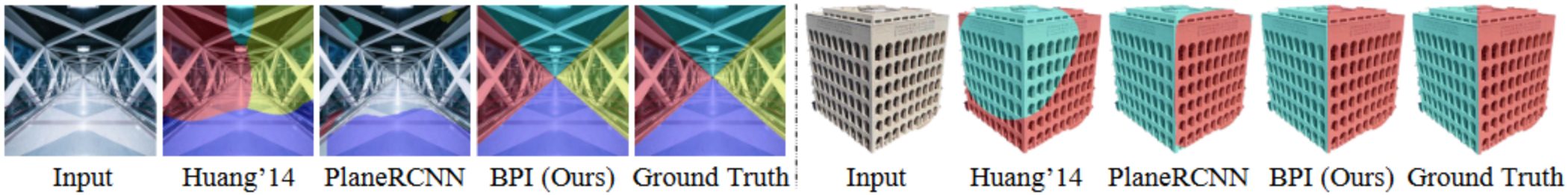


Figure 5: Visualization of the plane segmentation by different methods.

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).

Image Inpainting (Quantitiave Evaluation)

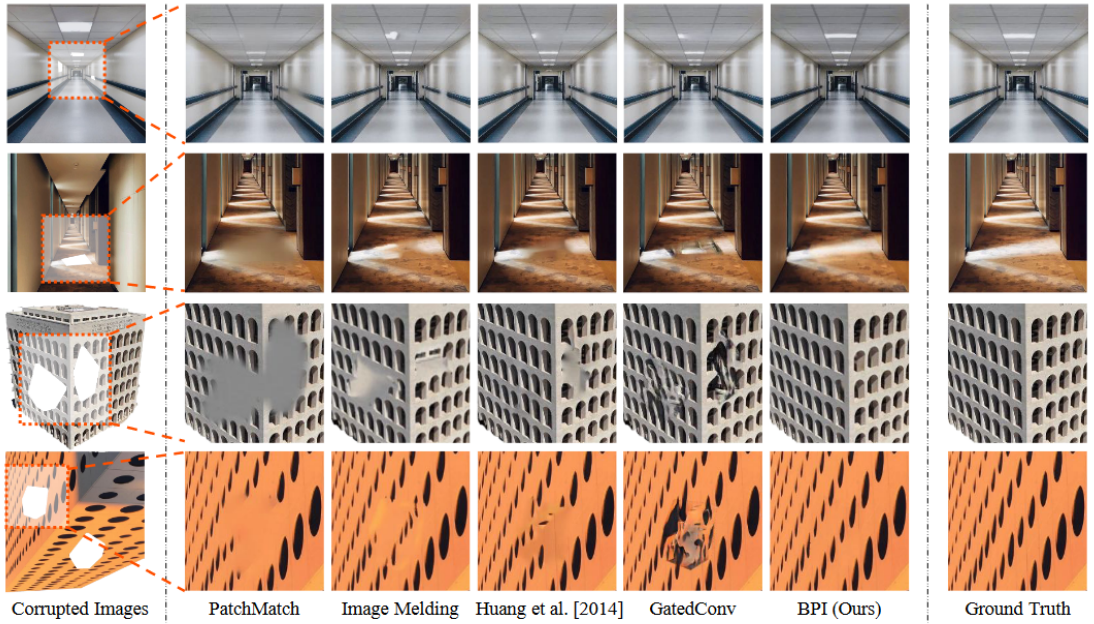


Figure 6: Qualitative results on the task of image inpainting. Compared with the baselines, our model can better preserve the regular structures even if they are sparse or have strong perspective effects.

	Corridors				Buildings			
Method	$\mathcal{L}_1$ 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.

Image Extrapolation (Qualitative Evaluation)

Figure 7: The "extrapolated" buildings. Content-Aware Scale, Kaspar et al. [18], 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. [16] fails to preserve the regular structure when inpainting large areas.

Randomly select 15 images and ask 15 participants to rank outputs.

- `61%` BPI (This Paper)
- `16%` Context-Aware Scaling
- `2%` Kaskar et al.
- `16%` InGAN
- `5%` Huang'14

View Synthesis (Qualitative Evaluation)

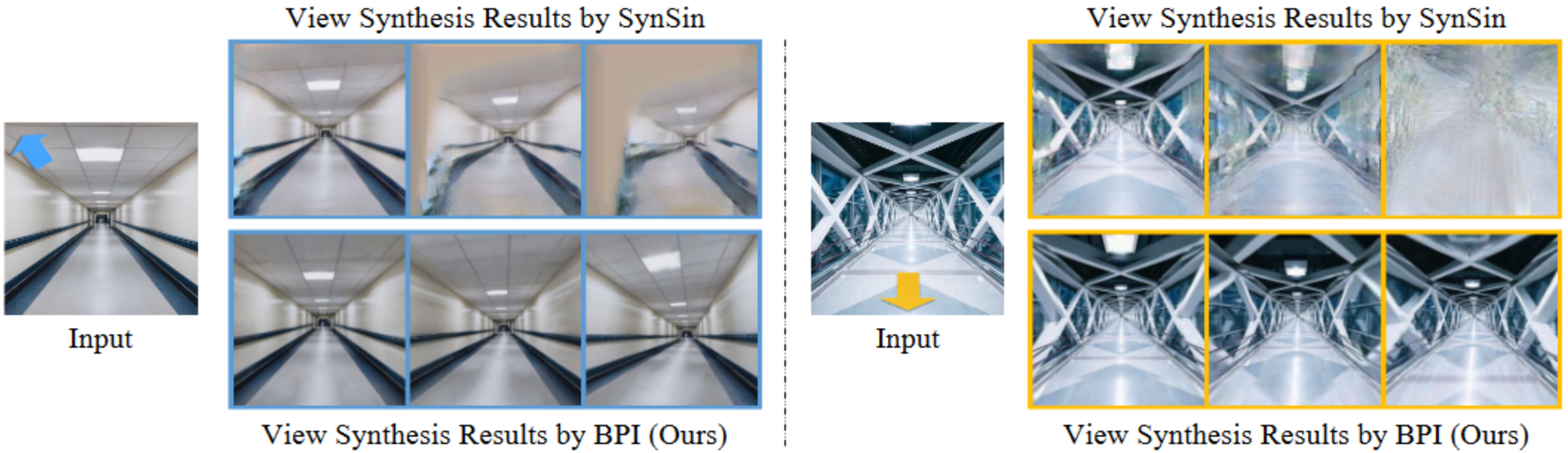


Figure 8: View synthesis from a single image of a corridor. Compared with the learning-based method SynSin, our model better preserves the regular patterns on the walls and also has remarkably fewer artifacts.

Randomly select 20 images from 3 trajectories and ask 10 participants to rank outputs.

- `100%` prefer BPI on Trajectory 1
- `94%` prefer BPI on Trajectory 2
- `99.5%` prefer BPI on Trajectory 3

Failure Cases

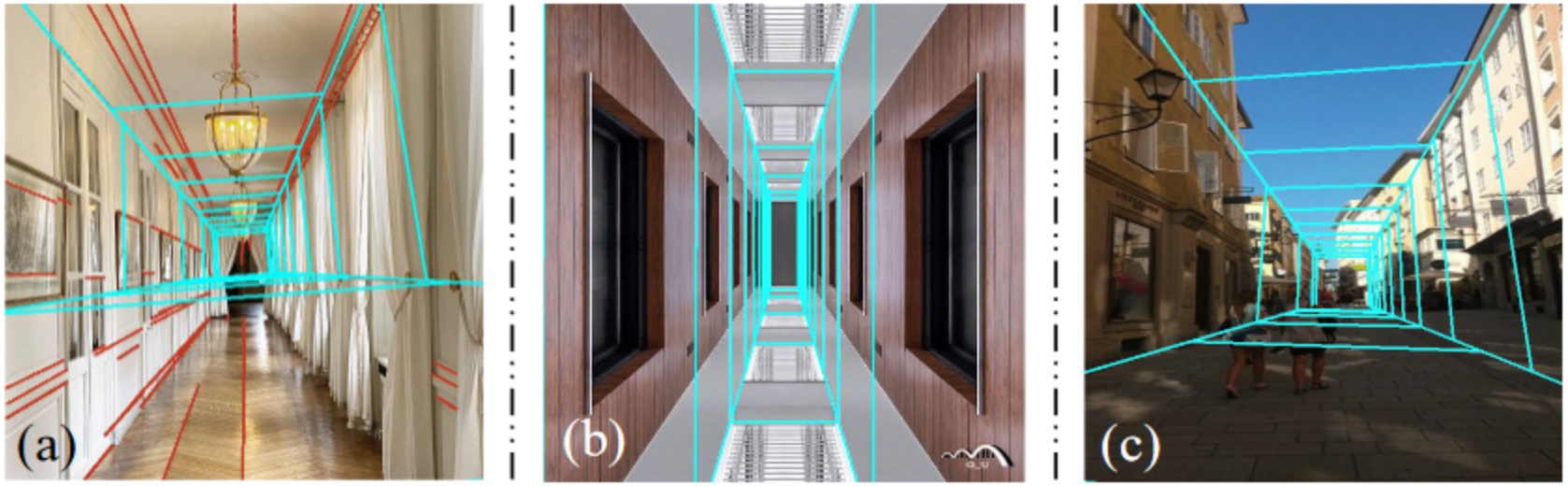


Figure 9: Failure cases. Our model may fail when neural networks misdetect visual cues (a). When image contains a solid color plane, our model may segment the pure white part to other planes (b). The inference might fail on irregular scenes, but could be mitigated with user interaction (c).

Failure Cases (?)

Interesting papers

- Perspective plane program induction framework
- The unreasonable Effectiveness of Deep networks as a Perceptual Metric.