PreSim: A 3D photo-realistic environment simulator for visual AI

Honglin Yuan[1] Remco C. Veltkamp[1]

***Abstract*—Recent years have witnessed great advancement in visual artificial intelligence (AI) research based on deep learning. To take advantage of deep learning, we need to collect a large amount of data in various environments and conditions. However, collecting such data is time-consuming and labor-intensive. Apart from that, developing and testing visual AI algorithms for multisensory models is expensive and in some cases dangerous processes in the real world. We present PreSim, a 3D environment simulator which provides photo-realistic simulations using a view synthesis module and supports flexible configuration of multimodal sensors to address both of these issues. For our view synthesis module we introduce novel depth refinement, adaptive view selection and layered rendering, to provide realistic imagery. We demonstrate that PreSim has several advantages: (i) it provides a photo-realistic 3D environment which allows seamlessly integrating multisensory models in the virtual world and enables them to perceive and navigate scenes, (ii) it has an internal view synthesis module which allows transforming algorithms developed and tested in simulation to physical platforms without domain adaption, (iii) it can generate a large amount of data for vision-based applications, such as depth estimation and object pose estimation.**

***Index Terms*—Simulation and Animation, Sensor Fusion, RGB-D Perception**

I. Introduction

RECENT years have witnessed great success of data-driven methods that use deep networks for computer vision tasks, such as depth estimation [1] and 6D object pose estimation [2]. These data-driven methods need a large amount of data to train and test their models. However, the process of collecting and labeling data is time-consuming and tedious. Gradually, the simulated environment is becoming an effective way to solve these problems, for it can provide amounts of annotated data for various AI tasks. A major current focus of environment simulators is to reproduce high-quality free-viewpoint rendering of real scenes. There are a number of open source simulators [3] to achieve this goal by parameter settings of scene details, including geometry, texture, lighting and 3D modeling of static objects. However, parameter setting is time-consuming and labor-intensive. Even with precise modeling and suitable parameter settings, the simulated world still lacks richness and diversity of the real world. This disadvantage may result in the failure of transferring algorithms that are developed and tested in simulation to physical platforms for many vision-based tasks, such as object recognition, obstacle avoidance, and visual navigation. This problem is known as the reality gap: the discrepancy between synthetic and real data.

Fig. 1: Snapshots from PreSim showing a robot moving in indoor environments: the left subwindow is a synthesized color image and the right subwindow is the corresponding depth map.

To address this issue, game engines which allow photo-realistic rendering have been leveraged to build virtual environments. However, the simulated environment heavily depends on the game engine's detailed datasets, which makes it impossible for users to build their own environments with their own datasets. On the other hand, game engines often use 3D graphics pipelines to provide real-time rendering. Thus, the rendering time increases linearly with the number of polygons to be rendered (scene complexity). To achieve real-time performance, it requires dedicated hardware and architecture design for 3D graphics. On the contrary, image based rendering which can provide real-time realistic imagery does not have these limitations. It only requires a sparse collection of captured images and allows a 3D scene to be visualized realistically

Manuscript received: September, 2, 2020; Revised December, 5, 2020; Accepted Feburary, 6, 2021.

This paper was recommended for publication by Editor Nancy Amato upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by CSC.

[1]Honglin Yuan and [1]Remco C. Veltkamp are with the Department of Information and Computing Sciences, Utrecht University, 3584CC Utrecht, The Netherlands h.yuan@uu.nl, R.C.Veltkamp@uu.nl

Digital Object Identifier (DOI): see top of this page.

without full 3D reconstruction. This approach has shown high-quality results in various environments [4]. In addition, the run time of image based rendering mainly depends on the display resolution of the output image rather than scene complexity.

Taking advantage of image based rendering, we introduce PreSim which is a 3D photo-realistic environment simulator for training and testing AI algorithms, as shown in Fig. 1. We aim to narrow the reality gap between simulation and reality by providing huge amounts of photo-realistic virtual RGB-D views from arbitrary locations for vision-based applications. The main contributions of our simulator are:

- A photo-realistic 3D virtual environment that provides users with ground truth poses of the multisensory model and free-viewpoint color-and-depth image pairs, even in regions where a global 3D reconstruction of the scene has inaccurate or missing data.
- A global visualizer providing real-time positions and whole trajectories of moving sensors, and a global 3D map.
- A sequence controller and recorder components to control the movement of sensors and store all the required information for developing AI algorithms.
- A novel view synthesis module built on image based rendering that combines depth refinement, adaptive view selection and layered 3D warping to lower the rendering complexity and improve the quality of synthesized images.

II. Previous work

Here we discuss several notable works in environment simulators and image based rendering that are closely related to our work.

Simulators. There are many environment simulators, such as Gazebo [3] and Atari [5], to model and visualize physical environments. Gazebo [3] is a well-known simulator that uses high-performance physics engines for rendering of indoor and outdoor environments. While Gazebo has rich features, it has limited abilities in creating visually rich environment of large scale and offer the realistic imagery. It has lagged behind various advancements in recent rendering techniques which allow photo-realistic rendering. A different class of approaches is based on game engines that enable rendering of photo-realistic camera streams [6], [7]. VRKitchen [6] is an interactive 3D Virtual environment built on Unreal Engine 4 (UE4). It can provide physically and visually realistic virtual kitchen environments. Habitat [7] uses Magnum engine to build photo-realistic virtual environments and provides a modular library for developing AI tasks (e.g., visual navigation) in it. However, they are limited by richness of simulated environments due to their high dependency on the engines. In contrast, our environment simulator enables users to build their own environments with their datasets.

More recently, public datasets such as SUNCG and Matterport3D have been used to create virtual environments. House3D [8] is based on the SUNCG dataset to provide 3D scenes of visually realistic houses. MINOS [9] provides access to the SUNCG and Matterport3D datasets and allows for environment configuration by removing or adding objects. Similar to our work, Gibson Env [10] uses image based rendering to provide photo-realistic rendering. While the goal of Gibson Env and our work is similar, Gibson Env requires a large amount of data to train a view synthesis network to avoid visual artifacts of synthesized images. A detailed comparison between our system and other environment simulators is summarized in Table I.

TABLE I: A comparison of PreSim to other environment simulators. 3D: 3D nature of the rendered scene, Photo-realistic: photo-realistic rendering, Novel: flexibility to be customized to other applications and Extendable datasets: permission for custom datasets.

Simulator	3D	Photo-realistic	Novel	Extendable datasets
Gazebo [3]	√		√	√
Atari [5]				
VRKitchen [6]	√	√	√	
MINOS [9]	√		√	
House3D [8]	√		√	
Gibson Env [10]	√	√		
Habitat [7]	√	√	√	
PreSim (ours)	√	√	√	√

Image based rendering. A thorough review of image based rendering methods can be found in [11]. The importance of maintaining the alignment of object boundaries between the color image and the depth map has been known in recent years [12], [13]. Ortiz-Cayon et al. [12] divide the image into superpixels to preserve object boundaries and then project each superpixel to the virtual view by a local shape-preserving warping to improve the blending quality. However, this approach does not consider photo-consistency and still suffers from silhouette flattening and inaccurate occlusion edges. There have been several works [13], [14] that improve the quality of synthesized images by filling holes. Nevertheless, the number of input views are fixed in these methods, which may lead to hole filling failure when the chosen views are useless or redundant. Instead, we use an adaptive view selection method to avoid such case.

In recent years, deep learning methods have been applied to produce novel views [15]–[17]. Hedman et al. [17] use a convolutional neural network (CNN) based architecture to estimate pixel weights for rendering. Flynn et al. [15] directly use a deep learning method for end-to-end view synthesis. However, in the current state, the end-to-end view synthesis methods [15], [16] still suffer from blurring and are not suitable for small datasets collected from a large range of viewpoints. In contrast, our view synthesis approach does not require dense image sets to train the model, and can render new views that have much larger changes with input views.

III. 3D photo-realistic environment

A. System overview

The architecture of our simulator is shown in Fig. 2. It is composed of a multisensory model, controllers, scene datasets, a view synthesis module and a global visualizer. Our simulator is based on Robot Operating System (ROS) which has a modular design and can be customized, upgraded and reused. In the virtual environment, we first import the point cloud of the real scene, which is generated from a 3D

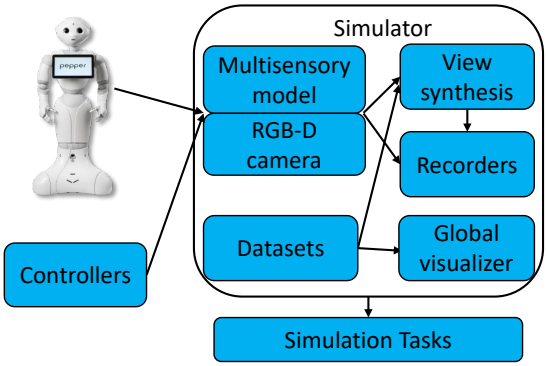


Fig. 2: The architecture of our simulator. It shows the main components of our simulator including a multisensory model, controllers, datasets, a view synthesis module and a global visualizer.

reconstruction, into the ROS and show it together with camera poses of input images in Rviz, a 3D visualizer for the ROS framework. Then we control the virtual camera's movement throughout the virtual world and estimate its 6D pose by ROS in real time. The estimated pose is then taken as a reference to select the most similar color-and depth image pairs in a query input dataset. Next, we use the selected color-and-depth image pairs to synthesize the virtual view based on our view synthesis module. At the same time, the whole trajectory of the moving camera and synthesized color-and-depth image pairs are logged. In the following, we provide more details on the individual components of our simulator.

B. View synthesis

Our goal is to build a free-viewpoint photo-realistic environment for vision-based tasks. Unlike previous methods that build the whole virtual environment on perfectly reconstructed 3D geometry, our view synthesis module takes a sparse set of RGB-D images as the input and produces new color-and-depth image pairs from arbitrary viewpoints. It consists of novel depth refinement and view selection steps followed by a fast rendering process. These components work together to lower the rendering complexity and improve the quality of synthesized images.

Depth refinement. Pixel-accurate alignment of object boundaries between color-and-depth image pairs and accurate depth values are necessary for high-quality rendering. This is because inaccurate depth values and misalignment often lead to various visible artifacts, such as ghost contours. During offline pre-processing, we introduce a pixel-to-pixel multi-view depth refinement algorithm to achieve this goal. We define the matching cost function $C(d_i)$ as,

$$C(d_i) = C_{pixel}(d_i) + C_{patch}(d_i), \tag{1}$$

where $C_{pixel}(d_i)$ and $C_{patch}(d_i)$ emphasize photo-consistency and edge preservation for the depth d_i of pixel i, respectively.

The photo-consistency $C_{pixel}(d_i)$ for the pixel i is measured by projecting it to other images, where we compare the color and gradient's similarities.

$$C_{pixel}(d_i) = \sum_{r \in R} \lambda ||x_i - x_r||_1 + (1-\lambda)|| \bigtriangledown x_i - \bigtriangledown x_r||_1, \tag{2}$$

where x_i is the RGB color of the pixel we calculate cost for in the target image and x_r is the resulting RGB color when the 3D point defined by depth d_i is reprojected into reference image r. R is the number of reference images which is selected by comparing the angle, distance and overlap it has with the target image. $||x_i - x_r||$ and $|| \bigtriangledown x_i - \bigtriangledown x_r||$ indicate the color and gradient differences, respectively. λ balances the influence of color and gradient terms and is empirically set. We set $\lambda = 0.9$ in all the experiments. Such a model has been shown to be robust to illumination changes and has the advantage of handling radiometric differences in the input images [18]. For a target image, we select ten reference color images based on distances and angles between the target and reference views. Next, we iteratively project pixels in the target image to the reference images and only save the cost value of the front-most pixel. In this way, we are able to avoid obtaining high cost values for correct depths.

The edge preserving term $C_{patch}(d_i)$ encourages the resulting depth map to have pixel-accurate alignment of depth with its corresponding color image.

$$C_{patch}(d_i) = \frac{1}{N} \sum_{q \in W_i} e^{-||x_i - x_q||_1}, \tag{3}$$

where W_i denotes a small (3×3) patch centered on pixel i and q is the neighbor pixel of i. N is the size of the patch (3×3) which is chosen empirically. $||x_i - x_q||$ computes the L1 norm between the RGB colors of i and q.

In the depth refinement process, we iteratively replace the depth value at a pixel with the one nearby that has the lowest matching cost. This is because similar pixels have low matching costs that are computed with the consideration of photometric and geometric relationship among pixels. The iteration starts with the top left pixel and traverses pixels in row-major order. After reaching the bottom right pixel, another iteration starts with the opposite direction. Four iterations are used in our experiments. The propagation is interleaved with the depth filtering. That is propagating good depth values to neighbors, if the costs are smaller than those of their neighbors. After propagation, we filter unusual depths with a weighted median filter [19] which is guided by the color image. The depth refinement algorithm is summarized in Algorithm 1.

Algorithm 1 Overview of the depth refinement procedure.

Input: Color images $I_1 ... I_R$, and depth maps $D_1 ... D_R$;
Output: Refined depth map D_1 for color image I_1 ;

1: Calculate photo-consistency cost C_{pixel} for I_1 and edge preserving cost C_{patch} for I_1.
2: Calculate matching costs $C = C_{pixel} + C_{patch}$.
3: Run propagation to update depth map D_1 and matching costs C.
4: Run weighted median filter.

View selection. The quality of synthesized images depends not only on correcting misalignment between color-and-image

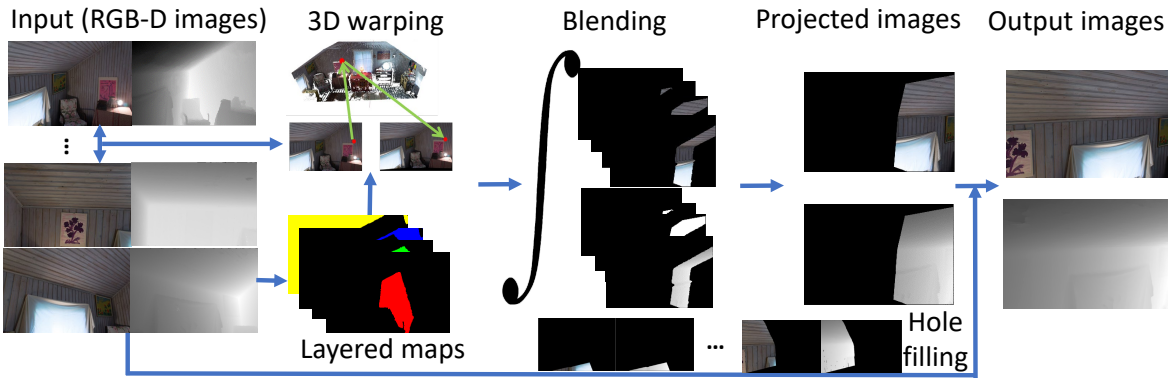


Fig. 3: Layered depth image based rendering. The input are RGB-D images. Based on depth values, we divide the depth map into layers. On each layer, we apply 3D warping to synthesize new images. Next, all the synthesized images are blended ($\int$) to produce color-and-depth image pairs. After that, we use other input images to fill holes in the synthesized image.

pairs [13] or filling holes [14], but also on selecting input views. Previous studies may choose incorrect or redundant views based on angles or distances between two views, which often leads to blurring images. In order to avoid such cases, we select input images considering not only angles and distances but also overlaps between two views.

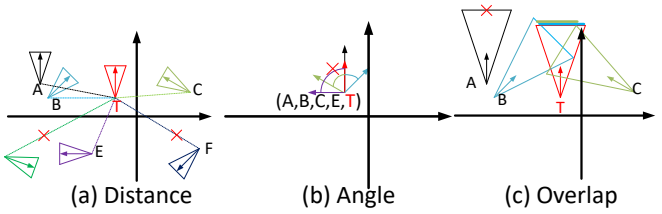


Fig. 4: View selection pipeline. (a) A, B, C, D, E, and F are input views and T is the target view. We first select a cluster of images having the smallest angles between their view vectors and that of the target. (c) Based on the overlaps between target and input views, we remove views having no overlaps with T.

Fig. 4 shows the selection process. Firstly, the distance between the input and the target views is calculated as shown in Fig. 4(a), where A, B, C, D, E, F are positions of input views and T is the target view. Then we rank the calculated distances and select the top ten images as a local group. From these local images, angles between the target and input views are calculated as shown in Fig. 4(b). If the angle is bigger than θ_{min} which is set to be the camera field of view, we remove it from the local input image group. Furthermore, in order to remove views, like A which has a small angle and distance, but no overlap, we calculate the overlaps (visible parts) between input and target views by projecting the input images into the target position. If the overlap is zero, we remove it from the local image group (Fig. 4(c)). To reduce the computation time of calculating overlaps, we downsample the input image with an equal sampling interval and only project sampled pixels into the target view.

Layered depth image based rendering. We propose to use layered depth image based rendering to synthesize new color-and-depth image pairs. The core part of image based rendering is 3D warping which projects pixels in the reference image plane to the world coordinate and then reprojects them to the new position in another image plane using camera intrinsic and extrinsic matrices. However, when objects in the background and foreground are projected to the same position, objects in the foreground may be occluded by objects in the background, which is caused by incorrect depth information or reprojection errors. To solve this problem, we evenly divide the depth map into layers based on the maximum and minimum depth values. On each layer, we apply 3D warping with corresponding color-and-depth image pairs to produce new images and then use a median filter with a 3×3 window size to fill missing information in each new image. After that, we blend these new images together to produce the final synthesized image. Since layered depths have the ability to represent occluded elements, our approach better handles the visibility problem. We found four layers to be a good trade-off between quality and speed.

After blending, the synthesized image may be constantly subjected to holes that are caused by the fixed number of input views used. To address this issue, we propose an adaptive view selection approach using a variable number of input images to fill holes in the synthesized image. We first project a key image selected based on the angle, distance and overlap it has with the virtual view, to the virtual position, and then detect holes in the synthesized image. If the size of the largest hole is bigger than a threshold (e.g., 0.04% of the whole image), we then choose another input image to fill holes. We iteratively run this process until the size of the largest hole is smaller than the threshold or the number of input views reaches the maximum which is set to be 10 in our experiments. The whole view synthesis pipeline is shown in Fig. 3.

C. Multisensory models and controllers

PreSim is designed to investigate the issue of domain transfer from simulation to the real world. Thus, it is important for the multisensory model to be constantly subject to constraints of space and physics such as collision and gravity.

Multisensory models. We use Universal Robotic Description Formats (URDF) to describe multisensory models (e.g.,

humanoid robots). Therefore, the model and its properties can be configured (e.g., types of sensors). As a demonstrator, we use the Pepper robot, which is a social humanoid robot from SoftBank.

Integrated controllers. We provide a set of practical controllers including joint state and navigation controllers to reduce the controlling complexity for the model's dynamic motions. The joint state controller is used to control the behaviors of joints of the model, including changing the pitch, roll and yaw angles. Our navigation controller allows controlling the model by directly sending movement commands. We also provide data recorders that allow saving all the data required by learning-based approaches. An example of a multisensory model and its trajectory is shown in Fig. 5.

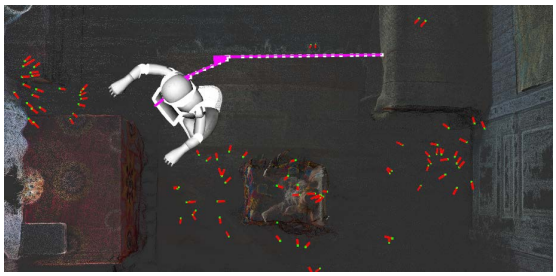


Fig. 5: The demonstrator model and its trajectory. Green points and red lines are positions and view directions of input views, small white squares and short pink lines are real-time positions and view directions of the virtual camera and the long line is the whole trajectory of the virtual camera.

IV. Experimental results

A. Evaluation of view synthesis

We evaluate PreSim on seven static datasets including our three own datasets (Study room, Table1 and Table2), four datasets (Attic, Dorm, Playroom, Reading corner) from [4], and two dynamic datasets (Ballet and Breakdancers) from [20]. There are less than 220 color-and-depth image pairs in the seven static datasets that contain black and texture-less objects (e.g., white walls and writing boards), reflective objects (e.g., bottles and lights) and objects with small geometric details. Each of the two dynamic datasets contains dancing people with a sequence of 100 color-and-depth image pairs, captured by eight static cameras which are positioned along an arc at 20-degree intervals.

Overall performance. We randomly choose a color image from the initial captured dataset as our ground truth image and then use other images to synthesize the chosen image. Fig. 6 shows some examples of synthesized color images and their corresponding ground truth images. Synthesizing one image (1280×720) takes $500 - 600ms$ on a computer with 6-core Intel Core i7 8700 3.19Ghz CPU. Even though [17] is faster than our approach which achieves 30 FPS, [17] is based on a GPU while our method is free from the GPU. From Fig. 6 we can see that our proposed method is able to provide high-quality synthesized images.

In order to quantitatively evaluate the free-viewpoint depth maps generated by PreSim, we also randomly choose a depth map from the initial captured dataset as our ground truth map and then use other depth maps to synthesize it. The root mean squared error (RMS) meters (lower is better) and average log10 error (lower is better) [21] are used to evaluate the rendered depth maps, and the experimental results are reported in Table II. As we can see, our approach achieves better performance on the static datasets. For static scenes, the error is mainly caused by initial captured depth maps. However, for the dynamic scene, it is more challenging, as it has more noise which is hard to be synthesized by input depth maps and the synthesized map has more errors in object boundaries when projecting input depth maps into the novel view.

TABLE II: Quantitative evaluation of rendered depth maps on different datasets.

	Table1	Attic	Playroom	Dorm	Breakdancers
RMS	0.426	0.472	0.483	0.501	2.419
log10	0.047	0.060	0.063	0.069	0.146

Comparison with other methods. We compare our method with state-of-the-art learning-based algorithms on Table1, Table2 and Study room datasets. The images in Table1 and Table2 are captured with a sufficient range of motion around different objects, as both datasets are designed for 6D object pose estimation. Study room is collected with sparsely captured images aiming to cover the whole room. The peak signal-to-noise ratio (PSNR) (higher is better) is used to evaluate image quality. Table III summarizes the quantitative evaluation results.

TABLE III: The PSNR comparison with different algorithms.

Methods	Average PSNR over 100 images (dB)		
	Table1	Study room	Table2
SM [22]	22.05	10.54	21.22
LLFF [16]	24.16	13.21	25.40
NeRF [23]	**37.98**	20.43	**37.81**
Ours	31.30	**26.59**	32.06

As we can see, even though the result of NeRF [23] is better compared to other methods on Table1 and Table2, our method achieves the best performance on Study room. This is because Table1 and Table 2 are densely sampled, while Study room is a sparse image set. It indicates that our method is more robust to the scene captured sparsely. Besides, our approach is free from training and can provide plausible synthesized images.

In Fig. 7 we compare our view synthesis method to Local Light Field Fusion (LLFF) [16] which also uses layered depth images. As we can see, while some unavoidable artifacts are visible in both methods, our approach provides generally sharper details and less noticeable artifacts.

In Table IV, we also compare our method with state-of-the-art algorithms which are designed for dynamic datasets. We use two reference images to synthesize a new image on ballet and breakdancers datasets. We can see that our algorithm performs the best on both datasets.

Effect of depth refinement. Fig. 8 shows example results of our depth refinement algorithm on different datasets. We can see that our method can significantly improve the quality of depth maps.

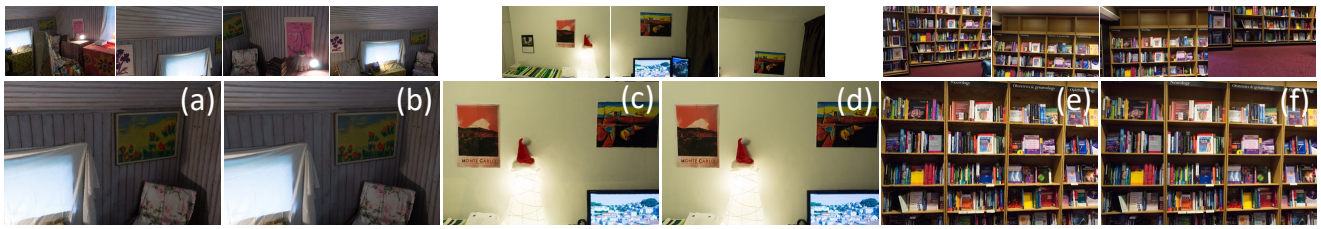


Fig. 6: Qualitative comparison of synthesized images ((a), (c) and (e)) with ground truth images ((b), (d) and (f)). The input images are in the first row.

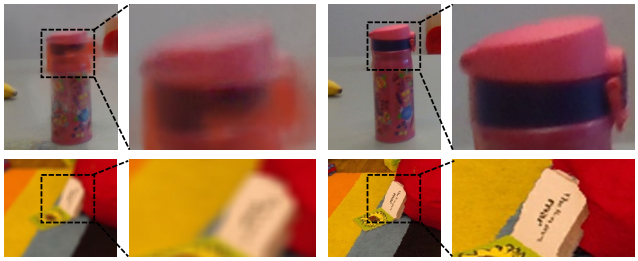


Fig. 7: Qualitative comparison of Local Light Field Fusion [16] (first and second columns) with our approach (third and fourth columns).

TABLE IV: The PSNR comparison with different algorithms.

Methods	Average PSNR over 100 images (dB)	
	Ballet	Breakdancers
VSRS [24]	30.23	31.17
Liu [25]	32.52	33.33
Dai [26]	32.55	31.77
Loghman [27]	30.36	31.64
Ours	**33.41**	**33.61**

We compare our depth refinement approach with the guided filter [28], a popular edge-preserving smoothing filter. We compare the average PSNR over 100 frames, as shown in Table V. While both approaches are able to improve the quality of synthesized images, our approach achieves better performance in various scenes.

Effect of adaptive view selection. The quality of synthesized images is influenced by the quantity of correct input images used. We compare hole sizes of synthesized images using different input views in Table VI. The number of input views used by previous approaches is fixed, which often results in big holes in the synthesized image (see Table VI), for the chosen views are not guaranteed to cover the whole virtual view. In contrast, our method with a variable number of input images can reduce the hole size significantly, especially when the virtual view is substantially different from input views. For example, for the Dorm dataset, the hole size is reduced by 18.05% and 5.82% compared to approaches with two and three input views, respectively. This is because the captured images in the Dorm dataset are sparser than those in the other datasets.

Fig. 8: The visualization results of depth refinement on Reading corner and Attic datasets.

TABLE V: The PSNR comparison between guided filter and our depth refinement.

	Average PSNR over 100 images (dB)	
	guided filter [28]	ours
Attic	29.39	33.29
Dorm	29.82	33.73
Ballet	28.85	33.44
Breakdancers	30.29	33.89
Reading corner	28.53	31.77

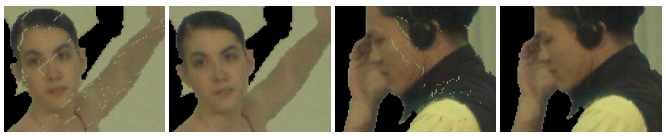
TABLE VI: Hole size comparison of the synthesized image using different input views. The hole size is defined by the percentage of missing pixels in the whole image.

	Hole size (%)			
	1 view	2 views	3 views	variable views (ours)
Attic	50.07	10.31	2.26	0.01
Dorm	60.19	18.35	5.85	0.03
Ballet	50.16	3.15	1.93	0.02
Playroom	45.96	10.51	6.21	0.03
Breakdancers	47.89	2.15	1.21	0.01
Reading corner	23.12	10.49	1.18	0.02

Effect of layered 3D warping. Fig. 9 shows some snapshots of synthesized images with and without layered 3D warping on two dynamic datasets which are more challenging than static ones. We can see that the background pixels are adequately removed and replaced by correct foreground pixels after layered 3D warping. In Fig. 10 we compare our method to the Z-buffer method [25] which is a commonly used approach to solve visibility problem. We can see that our layered 3D warping consistently improves the PSNR through all the testing frames.

B. Validation tasks

Depth estimation. Recently, deep learning methods have been used to predict depth maps for their corresponding color



(a) Ballet (b) Breakdancers

Fig. 9: Synthesized images without (first and third column) and with (second and fourth column) layered 3D warping on Ballet and Breakdancers datasets.

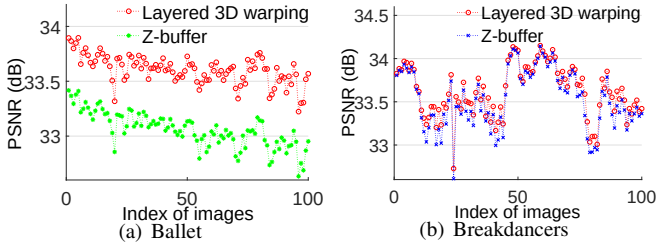
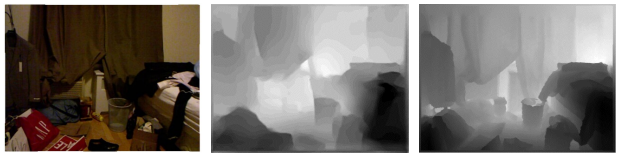


Fig. 10: The PSNR comparison with layered 3D warping and Z-buffer on each frame.

images. However, collecting real-world data is a tedious and labor intensive process. Compared with datasets produced by real-world data, synthesizing such a dataset requires less hardware, time and human labor while it is more likely to result in better quality. Taking advantage of PreSim, we use it to generate depth datasets which can be used as training data for learning-based depth estimation algorithms. We train DenseDepth [21], a popular network architecture for depth prediction on 30000 synthesized and 218 captured images. Then we test the trained model with color images that are never seen during training. Fig. 11(a) visualizes an example of the depth prediction on our testing dataset.

To prove the effectiveness of our simulator, we also test the trained model on a popular depth dataset, NYUDv2 [29]. Fig. 11(b) shows an example of the depth prediction result. We can see that knowledge learned from our simulated data

(a) Our testing dataset



(b) NYUDv2 testing dataset

Fig. 11: Qualitative results of depth prediction from DenseDepth [21]. The first column is the color image, the second column is the predicted depth map, and the third column is the ground truth map.

TABLE VII: Quantitative evaluation of the depth prediction in terms of RMS meters.

Dataset	Attic	Dorm	Playroom	Reading corner	NYUDv2
With PreSim	0.411	0.435	0.427	0.449	0.791
Without PreSim	-	-	-	-	-

can be transferred to real-world data in terms of accuracy, which indicates that our datasets can bridge the gap between simulation and reality. It is also verified by Table VII, where we evaluate the performance of depth prediction quantitatively based on the RMS meters. As can be seen, the model trained on real captured images fails, which is caused by the small size of the real captured data, while the model trained on synthesized data achieves good performance.

6D object pose estimation. We generate a benchmark dataset using our simulator for 6D object pose estimation. Based on our dataset, we organized the Shape Retrieval Challenge benchmark on 6D object pose estimation (https://yhldrf.github.io/Datasets.github.io/). The goal of this benchmark is to investigate how different state-of-the-art pose estimation approaches perform in terms of various object properties, including shapes, sizes, textures, changing light conditions, and occlusion.

TABLE VIII: The 6D pose estimation accuracy in terms of ADD.

	Banana	biscuit_box	chips_can	cookie_box
With PreSim	0.82	0.89	0.68	0.65
Without PreSim	-	-	-	-

(a) Qualitative evaluation of 6D pose estimation on the synthesized dataset

(b) Qualitative evaluation of 6D pose estimation on the real captured dataset

Fig. 12: Examples of 6D pose estimation results for different objects (banana, biscuit_box, chips_can, and cookie_box) with DenseFusion [2].

The DenseFusion network [2] is trained with 500 synthesized and 180 captured RGB-D images, respectively, and then tested with real captured images. The performance is evaluated by the average distance metric (ADD) (see Table VIII). A predicted pose is considered to be correct if ADD calculated with this pose is less than 10% of a model diameter. In Fig. 12 we also visualize estimation results. As shown in this evaluation, the model trained on real data also fails and the model trained on synthesized data provides accurate poses. It comes to the same conclusion as the depth estimation

experiments: knowledge learned from our synthesized dataset can be successfully transferred to real-world data without domain constraints.

V. Conclusion

We propose PreSim, a 3D photo-realistic environment simulator to develop vision-based algorithms for AI research. By leveraging a variety of indoor environment datasets and augmenting the data through a novel view synthesis module, we provide a large amount of data including photo-realistic color-and-depth image pairs with ground truth 6D poses. The generated data can be used for training and testing data-driven approaches for various AI applications such as depth estimation and 6D object pose estimation. Experiments demonstrate that our simulator narrows the reality gap between the virtual environment and the real scene. Thus, vision-based algorithms developed in the simulation can be transferred to real physical platforms without domain adaption.

Limitations and future work. Our depth refinement and view synthesis approaches are limited by the quality of the initial capture. If the captured depth map has too much missing information, our method is likely to introduce visual artifacts. For example, if less than 50% depth information of a transparent object is captured, PreSim is unable to generate accurate synthesized images for it. Besides, even though our trajectories used for synthesizing data contain a variety of movements like that of a person collecting data, new approaches are required to analyze the influence of different trajectory generation strategies on synthesizing data that is used for training depth/pose prediction networks.

References

[1] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *2016 Fourth international conference on 3D vision (3DV)*. IEEE, 2016, pp. 239–248.

[2] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, "Densefusion: 6d object pose estimation by iterative dense fusion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3343–3352.

[3] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.

[4] P. Hedman, T. Ritschel, G. Drettakis, and G. Brostow, "Scalable inside-out image-based rendering," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 1–11, 2016.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[6] X. Gao, R. Gong, T. Shu, X. Xie, S. Wang, and S.-C. Zhu, "Vrkitchen: an interactive 3d virtual environment for task-oriented learning," *arXiv preprint arXiv:1903.05757*, 2019.

[7] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik *et al.*, "Habitat: A platform for embodied ai research," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9339–9347.

[8] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian, "Building generalizable agents with a realistic and rich 3d environment," *arXiv preprint arXiv:1801.02209*, 2018.

[9] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun, "Minos: Multimodal indoor simulator for navigation in complex environments," *arXiv preprint arXiv:1712.03931*, 2017.

[10] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, "Gibson env: Real-world perception for embodied agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9068–9079.

[11] H.-Y. Shum, S.-C. Chan, and S. B. Kang, *Image-based rendering*. Springer Science & Business Media, 2008.

[12] R. Ortiz-Cayon, A. Djelouah, and G. Drettakis, "A bayesian approach for selective image-based rendering using superpixels," in *International Conference on 3D Vision-3DV*, 2015.

[13] G. Chaurasia, S. Duchene, O. Sorkine-Hornung, and G. Drettakis, "Depth synthesis and local warps for plausible image-based navigation," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 3, p. 30, 2013.

[14] J. Lei, C. Zhang, M. Wu, L. You, K. Fan, and C. Hou, "A divide-and-conquer hole-filling method for handling disocclusion in single-view rendering," *Multimedia Tools and Applications*, vol. 76, no. 6, pp. 7661–7676, 2017.

[15] J. Flynn, I. Neulander, J. Philbin, and N. Snavely, "Deepstereo: Learning to predict new views from the world's imagery," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5515–5524.

[16] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar, "Local light field fusion: Practical view synthesis with prescriptive sampling guidelines," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–14, 2019.

[17] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pp. 1–15, 2018.

[18] T. Brox and J. Malik, "Large displacement optical flow: descriptor matching in variational motion estimation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 3, pp. 500–513, 2010.

[19] Z. Ma, K. He, Y. Wei, J. Sun, and E. Wu, "Constant time weighted median filtering for stereo matching and beyond," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 49–56.

[20] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," in *ACM transactions on graphics (TOG)*, vol. 23, no. 3. ACM, 2004, pp. 600–608.

[21] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," *arXiv preprint arXiv:1812.11941*, 2018.

[22] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely, "Stereo magnification: Learning view synthesis using multiplane images," *arXiv preprint arXiv:1805.09817*, 2018.

[23] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *arXiv preprint arXiv:2003.08934*, 2020.

[24] Software for view synthesis, "Software for view synthesis," http://www.fujii.nuee.nagoya-u.ac.jp/multiview-data/mpeg2/VS.htm.

[25] J. Liu, C. Li, X. Fan, Z. Wang, M. Shi, and J. Yang, "View synthesis with 3d object segmentation-based asynchronous blending and boundary misalignment rectification," *The Visual Computer*, vol. 32, no. 6-8, pp. 989–999, 2016.

[26] J. Dai and T. Nguyen, "View synthesis with hierarchical clustering based occlusion filling," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 1387–1391.

[27] M. Loghman and J. Kim, "Segmentation-based view synthesis for multi-view video plus depth," *Multimedia Tools and Applications*, vol. 74, no. 5, pp. 1611–1625, 2015.

[28] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 6, pp. 1397–1409, 2012.

[29] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from RGBD images," in *European conference on computer vision*. Springer, 2012, pp. 746–760.