Reconstructing 3D Indoor Scene from RGB Equirectangular Panorama Images with Convolutional Neural Network (CNN) System

by

Chan Cheuk Pong

Temporary Binding for Examination Purposes

A thesis submitted in partial fulfilment of the requirements for
the Degree of Master of Philosophy
at The University of Hong Kong.

September 2021

Abstract of thesis entitled

Reconstructing 3D Indoor Scene from RGB Equirectangular Panorama Images with Convolutional Neural Network (CNN) System

Submitted by

Chan Cheuk Pong

for the degree of Master of Philosophy

at The University of Hong Kong

in September 2021

Constructing 3D virtual environments is an indispensable process in Virtual Reality application development because they define everything the VR users can see in VR worlds and can immerse players and induce the feeling of presence inside the VR environment. However, creating Virtual environments consume a lot of time and effort, even for professional 3D modelers. It is therefore important to explore methods to automate the generation of 3D scenes used in VR. Inspired by the rapid development of Deep Neural Network, specifically Convolutional Neural Network, we propose to accelerate the development cycle of virtual environment generation by a Deep neural network-based 3D reconstruction system, which takes 360 indoor RGB equirectangular panorama images as input, and outputs the generated 3D scene. Every 3D scene is enclosed with the room's 3D layout and is populated by 3D objects that exist in the input image. To simplify the virtual environment generation problem, we divide the entire system into 4 main subtasks, namely Room Layout Estimation, Object Detection, Object Pose Estimation, and Artistic Postprocessing. In each

chapter of this thesis, we introduce the background of each subtask, its related works, and how we propose to solve it. We also evaluate our approach by conducting quantitative or qualitative experiments. According to the results, most of the submodules can successfully operate on an error that is either competitive to relevant methods, or even better than existing approaches.

Throughout the chapters, we also explain our secondary contributions. We propose a visual representation enhancement algorithm to a room layout estimation network. Evaluation shows our approach can improve generated layouts' objective visual realness and optimize framerate and memory usage if the environment is rendered in Unity. Furthermore, we introduce our new object pose estimation dataset, created by taking inspiration from each of the previous related datasets and combining their unique advantages into our own one. We statistically show that our dataset is superior to existing ones in terms of image source diversity and richness of annotation features. Finally, we are going to cover the Unity-based Annotation Tool we created to accompany our dataset. It serves the purpose of reading and editing annotations and giving users the ability to annotate their custom datasets with 3DOF pose.

Word Count: 366 words

Declaration

I declare that this thesis represents my own work, except where due acknowledgement is made, and that it has not been previously included in a thesis, dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualifications.

Signed . . . .  . . . .

Chan Cheuk Pong

Table of Contents

Chapter 1: Introduction

1.1 Background and Motivation

In the Virtual Reality field, creating digital 3D environments is a fundamental and crucial process, because they must be used in most VR applications such as virtual professional job training, virtual architectural design tours, and video games. 3D virtual environments define what users can visually perceive in VR applications and possess a major role to immerse users into VR worlds. According to the style and nature of the applications, virtual environments are usually tailor-made by professional 3D modelers. However, these 3D scenes are created over the course of a few weeks, or even months depending on the scale of the VR application and the size of the workforce. In the long term, it is necessary to make virtual environment generation more intuitive and available to the public.

With the rapid development in Computer Vision and Machine Learning, modern researchers utilize deep neural networks to analyze large amounts of data, and train them using statistical methods to fulfill various tasks. For instance, Convolutional Neural Networks (CNN) excel in extracting visual features, such as edges and corners, from images and accomplish tasks such as Object Detection and Object classification. Moreover, neural networks are highly versatile. For example, the concept of Transfer Learning enables the same Object Detection Neural Network structure to be fine-tuned to detect different kinds of objects depending on the training data provided to it. This allows researchers to directly customize models that are previously trained on other sets of data, so that the final model can adapt to the new data researchers prepared.

In order to accelerate the process of creating 3D virtual environments, we propose a system consisting of DNNs that takes in 2D 360 indoors equirectangular panorama images and reconstructs the corresponding 3D environment, including the indoor room layout and the 3D objects within the room. To further elaborate, the system consists of 4 major parts: 3D Room Layout Estimation, Object Detection, Object Pose Estimation,

and Artistic Postprocessing. The first 3 parts of the system make use of DNNs models to achieve predefined tasks during the virtual environment reconstruction process. Customs datasets are crafted using resources online or by manual labor to train these Deep Learning models. To visualize the reconstructed environments, we output them in the Unity Game Engine, in which users can navigate the 3D scenes with built-in control schemes. The environments are also readily available for developing VR applications, since Unity supports native and 3rd-party VR development packages, as well as saving and loading entire generated environments.

1.2 Main Contributions

We propose 4 main contributions, namely the 3D reconstruction system, an Object Pose Annotation Tool based on Unity, an Object Pose Estimation Dataset, and an enhancement to HorizonNet that converts Point Cloud representation to Mesh representation. We explain our contributions as follows:

We propose a system of DNNs that converts a 360 RGB indoor equirectangular panorama image into its corresponding 3D digital environment. It supports non-cuboid fully textured room layout with mesh representation and a diverse set of static CAD mesh models as object representation. Indoor room layouts are generated with the inspiration from RNN-based HorizonNet. Objects are detected by a fine-tuned M-RCNN with enhanced versions of the 2D-3DS and the SUN360 datasets. Object poses are estimated by a fine-tuned PoseFromShape CNN. Training data comes from a new dataset we created, which will be covered below. Object retrieval is based on a semi-sphere projected view to ROI matching algorithm.

We propose a new Indoor Object Pose Estimation dataset which emphasizes on object poses seen in indoor living rooms, offices, and bedrooms. Categories range from large-scale furniture such as beds and tables, to smaller objects including pillows and bottles. Images and CAD model objects were collected online and used in the annotation process. Helpers from HKU and our research lab were chosen to participate in annotating images.

We propose a new Unity-based Annotation Tool GUI in order to assist Object Pose Estimation. Users can utilize the tool to align objects in the images with the 3D pose of the corresponding CAD model. Compared to other relevant existing datasets, our proposed dataset collects images from a larger number of online sources which can improve image diversity. We also have more 3D models overall, or per category, as opposed to previously proposed datasets.

We propose an enhanced method to display and represent HorizonNet indoor room layout output in terms of practical usability in VR applications. To be precise, we convert the original Point Cloud representation of the HorizonNet output to a 3D mesh representation through extracting the inferred data from HorizonNet. Mesh are dynamically generated again in the Unity Game Engine. This method ensures same layout accuracy between the two representations, while increasing the objective realness of the 3D layout. As experiment results also show, our mesh representation can improve framerate performance, as well as reduce the use of memory resources significantly.

1.3 Thesis Organization

The following sections of this thesis will be as follows.

In Chapter 2, we explain 3D indoor layout generation through HorizonNet, and the conversion from Point Cloud to Mesh representation in Unity.

In Chapter 3, we describe the process of detecting objects within 360 equirectangular panorama images from CNN, as well as the object retrieval from detected 2D object image to 3D CAD models.

In Chapter 4, we elaborate on our new Indoor Object Pose Estimation dataset, along with Pose Estimation of detected objects from panorama images through the use of CNN.

In Chapter 5, we cover the final steps of postprocessing data from all the CNNs and reconstructing the 3D scene.

Finally in Chapter 6, we conclude the thesis and include possible future works.

Chapter 2: Indoor 3D Layout Estimation

2.1 Introduction

To reconstruct a 3D scene from a 360 panorama image, it is necessary to estimate the 2D pixel location of walls, the floor and ceiling of a room from images, also known as 3D Room Layout Estimation. This topic has been in active research for over the past decade. Almost all of the previous approaches follow the Manhattan World Assumption, which states that all walls should be perpendicular to one another, and that the floor and ceiling should also be at 90 degrees against all walls. Furthermore, all layout surfaces should align with the 3 principal axes, according to the assumption [1]. Because of this important premise, the first solutions of the 3D layout estimation problem revolved around obtaining vanishing points from 2D images and generating layout hypotheses from them [2 - 4].

Along with the improvement in computing resources and the interest in Artificial Neural Network, researchers started to extract image features, such as edge maps, to assist in layout generation by matching vanishing lines with them [5 - 7]. Some approaches even estimate layout by only using extracted features using CNNs [8 - 12]. Apart from using RGB inputs similar to the above approaches, some previous works also use RGB-D images as input to compute indoor 3D layouts [13, 14].

In the context of Virtual Reality, we believe it is necessary for the generated layout to be complete, realistic and loyal to the original panorama input. Most of the previously mentioned methods take perspective images as input [2 - 6, 8, 10, 13 - 16], which have a low field of view and visually capture only a portion of the 3D environment. Environments generated from perspective images often only contain a subset of surfaces of the corresponding real-life scene. On the contrary, 360 panorama images have 360 degrees horizontal and 180 degrees vertical full range FOV, which thoroughly embeds all environmental data into a single RGB image. There are some approaches that use panoramas as input [7, 9, 11, 12, 17, 18]. Layouts that are produced under 360 panorama input ensure all surfaces' visual information are captured and analyzed in

the system. Consequently, a complete interpretation of the corresponding 3D scene can be created. In terms of the realness of generated 3D layouts, the material of surface texture is one of the most important factors. When displaying room layouts, depending on the system implementation, surface texture can be represented by a medoid color [10]. Compared with methods that extract image texture as surface texture [7, 8, 9, 11, 12, 17, 18], the environment produced will feel synthetic and unrealistic, breaking immersion as a result. Finally, we believe the created layout should stay structurally loyal to its input image. while some previous approaches only support cuboid structure [17, 19], some can estimate more than 4 walls and produce Manhattan aligned structure that is identical to the input regardless of the shape of the room [7, 9, 11, 12]. Having non-cuboid structured layout can improve the diversity of possible 3D layouts and make them more believable and natural.

In our approach, we are highly inspired by HorizonNet [12] and utilized its RNN to estimate 3D layout. Generated layouts can support complete non-cuboid layout, and fully textured surfaces. We also enhanced the representation of the layouts by converting Point Cloud surfaces to dynamically created mesh surfaces in the Unity Game Engine [27]. We evaluated the conversion process by setting up fly-through trials in both original point cloud layout and our mesh represented layout. Results show that our mesh representation is objectively more visually pleasant and realistic. In terms of performance, mesh representation also has overall more framerate than point cloud output. Rendering mesh on screen also consumes less memory resources than displaying point cloud environments.

2.2 Related Works

The problem of 3D layout estimation was initially tackled using geometrical information such as vanishing lines from the 3 principal axes. For example, a dynamic Bayesian network model was trained, inputting data corresponding to vanishing points, in order to estimate Wall-Floor boundary in each column of 2D images [2]. Some other approaches first calculate vanishing lines in the images and generate layout hypotheses based on the detected lines. The hypotheses were then ranked based on different

criteria, such as Orientation Map [3] and Geometric Context [4]. As mentioned above, most of the previous approaches generate layouts and construct estimation algorithms based on the Manhattan World Assumption [1]. Following the rapid advancements of ANN techniques, using FCN to predict edge maps became a popular method as part of the room layout estimation process. The generated edge maps were applied, instead of previously used OM and GC, to rank layout hypotheses created by vanishing line sampling [5, 6]. As time progressed, layout estimation methods without the consideration of vanishing points started to appear. It is worth noting that many of these approaches utilize an encoder-decoder network structure to construct their CNNs, regardless of the input image type [7, 8, 11]. Another similarity between these methods is that they use CNNs to detect corner keypoints in input images in order to construct layout hypotheses.

Despite the collective approaches to estimate 3D layouts from RGB images, only a small subset of methods are truly relevant to layout reconstruction in the context of Virtual Reality applications. For a VR user, it is not acceptable to only see a portion of the entire indoor environment, such as only two walls and a floor, because this limits the users' ability to explore the whole 3D scene. Only with 360 panorama input can layout estimations be complete, because of its full FOV property. An example of 360 images as input is PanoContext [17], which applies previous OM [3] and GC [4] ranking techniques to panorama images. Another example is LayoutNet [7], which uses predefined vanishing lines to predict panorama edge maps and corner keypoints in order to create 3D layouts. Another important factor that affects the VR users' experience is the structural detail of the estimated layout. In most cases, indoor rooms are of different shapes. However, some previous approaches only support cuboid layout outputs [17, 19], which hinders the diversity of possible room shapes and does not reflect the real structure of the indoor environments. VR users would feel bland and bored if they were to explore a 3D box with reattached texture for every environment. The final contributing factor towards a satisfactory reconstructed 3D layout, is the realness of the 3D scene, as realistic environments greatly induce users with the feeling of presence [20], which in turns triggers the feeling of immersion [21]. Some previous research has successfully achieved that through extracting surface texture directly from

the input panorama image [7, 8, 9, 11, 12, 17, 18]. Using the original surface color of the input can make user feel as if they are truly inside the generated environment, instead of having single-colored fake walls surrounding them.

Among all the aforementioned methods, only LayoutNet [7], DulaNet [9], CFL [11], and HorizonNet [12] have the suitable premise to generate 3D layouts for VR applications. Specifically, HorizonNet is the most inspiring with its state-of-the-art estimation performance due to its novel 1D-data representation and Pano Stretch Data Augmentation techniques. Therefore, our approach heavily utilizes its layout prediction capabilities to generate non-cuboid room structure.

2.3 Methodology

2.3.1 HorizonNet

In the first stage of our 3D reconstruction system, we utilize HorizonNet [12] to generate 3D layouts. HorizonNet has a few main technological breakthroughs that made it one of the state-of-the-art approaches in the field of 3D layout estimation.

2.3.1.1 1D Layout Data Representation

As mentioned in the last section, many recent methods utilize CNNs to predict 2D edge maps [5, 6] and keypoint heatmaps [7, 8] to construct room layout hypotheses. However, these data structure are of O(HW), which means the data spans across the whole 2D image, occupying Image Height x Image Width in both memory space, and the time complexity when analysing the 2D maps. HorizonNet improves the data representation drastically by only storing the pixel location of floor-wall boundary, and ceiling-wall boundary for each column. Including another 1 x Image Width vector representing the existence of layout corners, the resulting output of the network is only 3 x 1 x Image Width (O(W)), which is a huge advancement compared to typical 2D feature map extraction approaches.

2.3.1.2 Network Structure incorporating CNN and RNN

HorizonNet first used a CNN to extract input image features, then employed a RNN to estimate room corner existence, ceiling-wall boundary, and floor-wall boundary for each column. Regarding the feature extraction CNN, a ResNet-50 [22] was adopted and feature maps extracted in each ResNet block were convoluted, upsampled and concatenated, in order to combine low-level and high-level features. The resulting feature map will be sent to a LSTM [23], a variant of RNN that can operate prediction based on previous predicted result. This property is particularly beneficial for layout estimation, because 2D corner positions in images are naturally correlated with each other, so even though corner points are occluded in input panorama image, LTSM can still infer corner location accurately. Specifically, since layout surface boundaries are essentially connected, strong relation exists between neighboring column boundary position, so HorisonNet adopted a bidirectional structure [24] to strengthen the prediction accuracy.

2.3.1.3 Pano Stretch Data Augmentation

Data augmentation is necessary when original training data is too deficient and causes machine learning models to overfit the data. Currently, datasets that contain indoor panorama images are scarce, with SUN360 [25] and 2D-3D-S [26] being one of the rare examples. To augment panorama inputs, HorizonNet effectively applies scaling to ground truth 3D layouts and reflects uv position changes in panorama images. Specifically, its approach first map each pixel coordinate in equirectangular image into their corresponding spherical angular yaw/pitch coordinates. Angular data are then converted to 3D xyz coordinates and are scaled towards x or z axes directions. The new xyz coordinates are then reprojected back to angular and finally, newly modified pixel coordinates. Data is augmented on the fly during the training process via this algorithm. As a result, the network's estimation performance is boosted when the pano stretch data augmentation is employed.

2.3.2 Conversion of Layout Representations from Point Cloud to Mesh

2.3.2.1 Point Cloud and Mesh Representations

Originally, HorizonNet's default built-in display function represents generated 3D layouts as Point Clouds. Point Clouds are often used to display 3D environments due to intuitiveness to record virtual environment data. A single point in the point cloud only needs to store its spatial location xyz coordinates, and its RGB color. Individual points group together to form a complete environment, while users can customize the point density and point size to suit different visualization demands. Although Point Cloud representations has high flexibility and straightforward data structure, we believe that it should not be used in Virtual Reality applications. Since individual points in point clouds are separated spheres or circles depending on the rendering method, gaps between points often exist no matter the point size and density. Of course, a higher point density or overlapping large sized points could diminish the gaps, but either more computing resources would be required to render dense environments, or big dots in surfaces will be visible to users, harming the visual appearance of the 3D scene and its realness. Therefore, it is necessary to adopt a better layout representation for VR applications.

We selected 3D mesh as the new representation medium for 2 distinct reasons. Mesh surfaces are formed using interconnected polygons such as triangles. Unlike Point Cloud, mesh surfaces do not have spatial gaps and holes within them, so they look significantly visually pleasant in comparison. Moreover, we are implementing the 3D reconstruction system in the Unity Game Engine [27]. Since generating dynamic mesh surfaces is natively supported in the engine, adopting mesh surfaces can optimize both the implementation process, as well as practical mesh generation performance.

2.3.2.2 Converting Room Layout Representation

To generate the 3D layout outputted by HorizonNet in mesh form, we first intercepted its inferred data output from the original system. Specifically, we instructed our system to trigger HorizonNet's inference algorithm after receiving the panorama image input. Then, room corners xyz coordinates, point cloud density, surface masks, and surface

pixel RGB values were extracted from HorizonNet's post-processing module. The data is then transferred from HorizonNet to our Unity-based system. However, because of the fact that the original output data, such as texture array, are NumPy arrays which cannot be directly feed into Unity C# programs, we used an intermediate .json file to store all to-be-transferred data.

After all data from HorizonNet were received, we utilized the extracted corner coordinates to establish dynamic mesh planes for our Unity-based layout. Typical Unity mesh are rectangular planes or triangles, which can be applied to generate walls. However, as floor and ceiling surfaces are often non-cuboid, we adopted a custom triangulator [28] and successfully recreated surfaces identical to the original Point Cloud version. For texture binding, we used the corner coordinates and point cloud density to calculate the texture dimension of different surfaces. After that, we masked the rectangular texture with mask arrays obtained from HorizonNet in order to prevent incorrect texture binds due to non-cuboid surfaces. Finally, we assign the new textures to their respective surfaces and save all the meshes and textures created as Unity prefab objects, to ensure the generated layout can be reused by loading it during engine runtime.

To further explore the produced layout, we can import a third-party Mixed Reality plugin, VotanicXR [29] and instantiate a VR character in the scene for users to visually perceive the layout.

2.4 Experiments

2.4.1 Layout Preparations

To evaluate the conversion from Point Cloud to Mesh representation, we conducted a fly-through experiment under various layout structures, point cloud density and representations. To be precise, we first converted all the test set images from HorizonNet -- the 166 test set images from LayoutNet [7] and 65 extra layouts that

were re-annotated as non-cuboid layout, from Point Cloud to intermediate .json format for mesh conversion. We also saved two distinct Point Cloud layouts for each panorama image under different point density. The first one is 80 points-per-meters (ppm), which is the default density set by HorizonNet. 80ppm is also the point density we used to create mesh structures. Therefore, 80ppm Point Cloud layout is similar to its mesh counterpart in terms of layout rendering performance. Another configuration is 300ppm, which is significantly larger than the default density. We select this density because theoretically, higher point density results in a higher perceived quality [30], which may perform similar to mesh representation in terms of layout visual appearance. Point size are scaled according to the point density to visually minimize the gap between points. Specifically, we set 0.015 for 80ppm layout, and 0.003 for 300ppm layout. Point Cloud layouts are saved in .ply format for importing into Unity. 3 panorama images are chosen from the 65 re-annotated set for trials based on the size, dominant color palette, and shape of the generated non-cuboid layout (Fig. 1a).

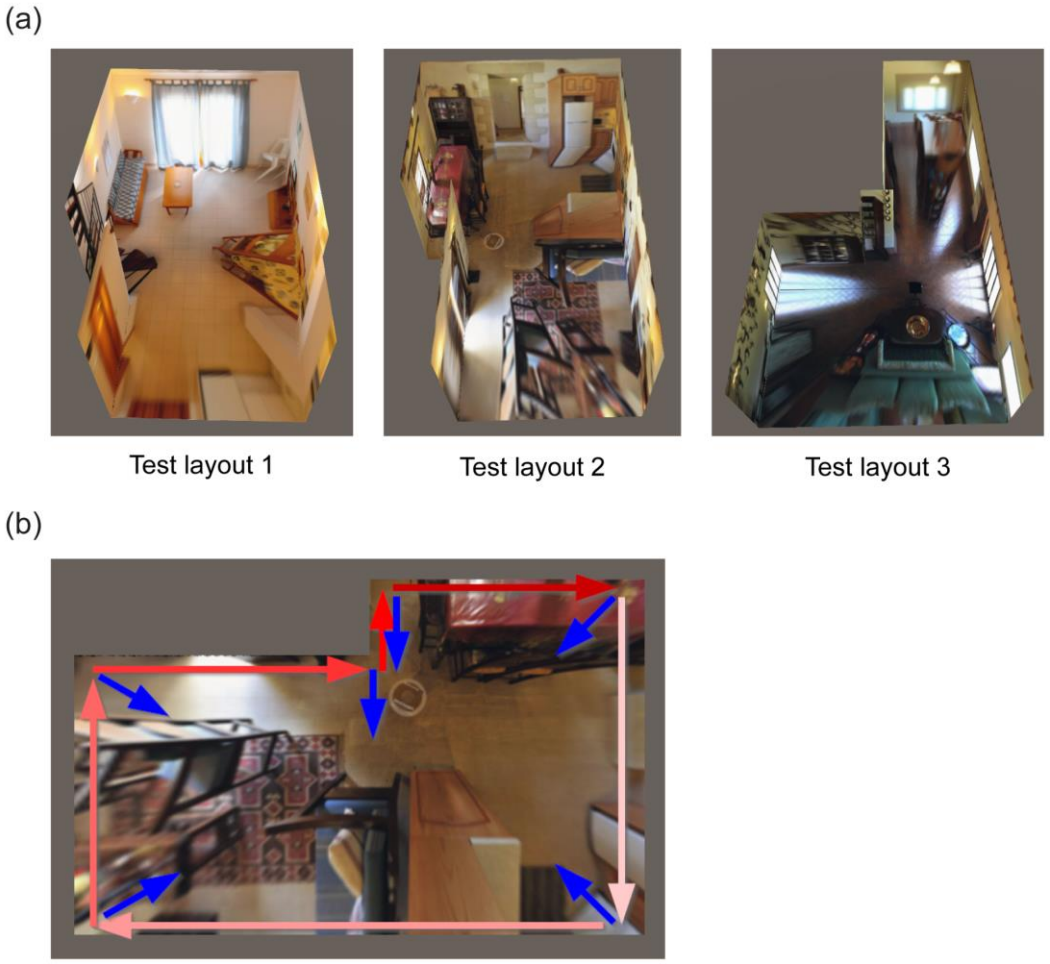

Fig. 1: (a) The three test layouts for the room layout experiment (b) example of a traverse path of the VR character. Blue arrows: checkpoints; Red arrows: General movement direction

2.4.2 Procedures

As each chosen image had 3 distinct 3D layouts generated under different representation and point density, layouts from the same type were grouped into a single Unity 3D scene. For example, scene 1 only contained the mesh representation 3D layouts from the 3 chosen input images, while scene 2 only consisted of 80ppm layouts of the chosen panorama images. Layouts are placed evenly spaced across the scene, and we imported a third party VR plugin VotanicXR [29] to traverse the virtual layouts with a VR player character. For each test layout, we predefine a static path for the VR

character to travel on (Fig. 1b). Red arrows represent the general clockwise path, and blue arrows represent “Checkpoints”, where we instruct the VR character to stop and face towards the open area of the environment, forcing the engine to render most of the layout on screen. After the character finishes traversing a single layout, it will be teleported to another layout, until it finishes its journey in all 3 layouts. Such an experiment took place 3 times, as there were 3 distinct scenes for each representation configuration. During the VR character’s journey through the layouts, frame rate, memory usage will be recorded in real time for evaluation.

2.4.2.1 Hardware Specifications

The experiment was conducted with a head mounted display (VIVE Pro; HTC, New Taipei City, Taiwan) with a 110° horizontal FOV. The HMD is connected to a 64-bit Windows 10 PC with Intel® CoreTM i7-7700K CPU @4.20 Ghz processor, 16 GB RAM, and NVIDIA® GeForce® RTX 2080 Ti graphics card. Layouts were rendered with SteamVR (Valve, Bellevue, WA, United States) and developed in Unity 2018.4.6f1. This specification is also stated in our published Springer poster [31].

2.5 Evaluation and Results

2.5.1 Evaluation Methods

For evaluating the conversion from Point Cloud to Mesh representation, 3 evaluation metrics are adopted - Objective Realness, frame rate performance, and memory usage performance. The choice is explained as follows:

2.5.1.1 Objective Realness

As previous research shows, the more realistic the virtual environment, the more presence and immersion it can induce to VR users [20, 21]. This is the most important reason behind the idea to convert point cloud layouts into mesh, as we expect representing 3D layouts with mesh is visually more appealing than using point cloud. Therefore, we would like to compare objectively how realistic the newly converted mesh representations are. Specifically, we capture a small portion, preferably one of

the objects shown on a surface, from the 3 different layout types of the same input image. That way we can observe the difference between representations clearly.

2.5.1.2 Frame Rate and Memory Usage Performance

Overall Frame rate when executing VR applications is an important consideration because a low frame rate can lead to simulator sickness [33] and decreased presence [32]. We generally also would prefer a lower usage of memory, because more operations including programming logic and simulations can be run simultaneously. Point Cloud and Mesh representation has drastically different principles of formation. Point Cloud is formed from individual independent color points, while mesh is formed by interconnected polygons. As a result, the computing resources required to render the two can be different. As mesh is the default way of displaying objects in the Unity Game Engine, we expect rendering mesh in Unity is the most memory-efficient and takes the least time to render on screen. Therefore, we predict that the frame rate is also higher than Point Cloud. We utilize a Unity statistics monitor package [34] to record frame rate in frames-per-second (fps) and memory usage in Megabytes (MB) during the whole experiment in real time.

2.5.2 Results

Fig. 2: Different representations of the same object inside the generated layouts.

2.5.2.1 Objective Realness

We first discuss the visual layout results obtained with the 3 representation configurations. Fig. 2 shows a portion of the rendered layout in mesh, 80ppm point

cloud, and 300 ppm point cloud respectively. We can clearly notice that the mesh output does not have any seams and artifacts within it. However, 80 ppm point cloud layout, which has the same texture point density with mesh, suffers from heavy noise in the texture. This is due to the large point size used to compensate for the gaps generated from the small point density. Consequently, points are quite obvious and apparent even to bare human eyes. However, when we observe the 300ppm point cloud sample and compare it to its mesh counterpart, we notice there is not much difference. Gaps or incomplete surfaces are non-existent to the eyes. Therefore, we can conclude that mesh and 300ppm point cloud layouts are the most favorable in terms of visual appearance and realness, with 80ppm point cloud being less visually convincing.

2.5.2.2 Frame Rate and Memory Usage Performance

Fig. 3 illustrates the fly-through experiment results. Fig 3(a, b, c) visualize the frame rate performance in the 3 test layouts generated from the chosen input panorama images respectively. Each subplot has 3 distinctly colored lines which indicates the type of the generated layout. The last subplot shows the overall mono memory usage throughout the VR character's whole journey in the 3 test layouts. The 3 lines presented in the subplot again reassemble the different types of layout representations.

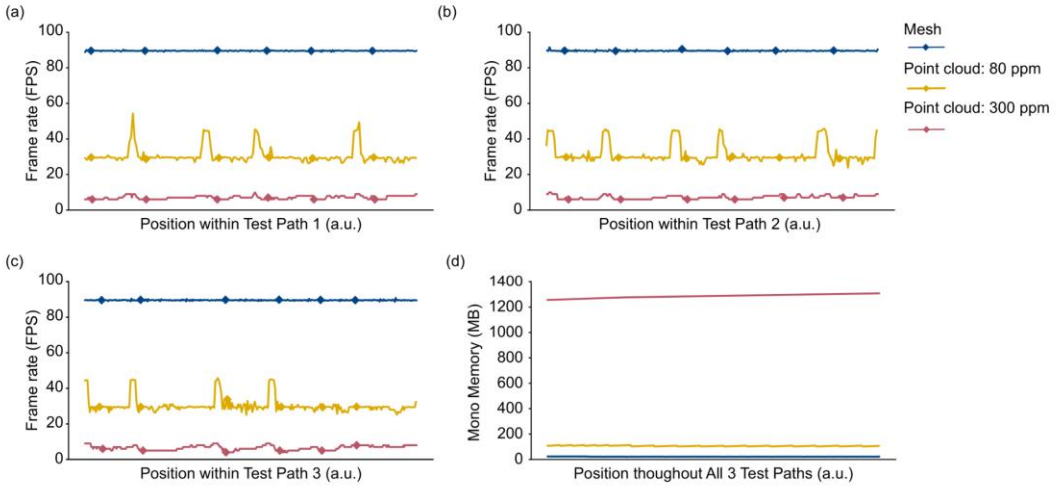

Fig. 3. Frame rate and memory usage in different representations

Throughout the 3 test layouts, we can see that the mesh layout has significantly higher frame rate, with an average of almost 90 frames per second. Performance is also stable in comparison. 80ppm point cloud output has the most fluctuating results, with a mean fps of 30. An interesting fact is that frame rate tends to spike right before a checkpoint. We suspect this is because every checkpoint is close to a corner of the room layout. So as the VR character approaches one, the camera renders the least amount of layout before it turns around to face the open area. Although 300ppm point cloud layouts have fewer fps spikes, they have the worst frame rate performance, with a mean fps of around 7. It is worth noting that the frame rate performance is consistent for all test layouts, which implies that the size, shape, and color scheme of the resulting 3D layouts do not affect the high fps performance of mesh representations.

In terms of memory usage, we again notice mesh representations consume the least mono memory, with only an average of 21.9MB used for rendering. 80ppm point cloud uses slightly more, which is as expected as their texture point densities are the same. 300 ppm point cloud layouts consumed the largest amount of memory, with over 1200MB used only on rendering the dense point cloud.

Our experiments show that our mesh representation of HorizonNet layouts is both memory efficient and lightweight enough for smooth rendering. Furthermore, mesh layouts are also visually more appealing to the eyes compared to point cloud representations.

Chapter 3: Object Detection

3.1 Introduction

Detecting objects within an image has always been a critical and fundamental topic in the field of computer vision. Its solutions have truly evolved over the past 20 years. In this section we will cover how past methods have dealt with image object detection.

Before Artificial Neural Network was popular, researchers combined traditional computer vision techniques such as feature descriptors [35, 36] along with classical machine learning algorithms [37, 38]. In the early stage of machine learning, these solutions were sufficient to tackle computer vision tasks such as image classification, object detection, and face recognition [39].

Artificial Neural Networks are systems that utilize a series of nodes called neurons, to analyze the relationship between its input and output data. Unlike feature descriptors, parameters inside neural networks can be statistically optimized to adapt to specific datasets, such that it can predict outputs accurately, even with complex inputs. Although there are many types of ANNs, the use of CNNs to accomplish computer vision tasks became prevalent in the last 10 years, with AlexNet [40] being the first pioneers. The architecture of AlexNet contains layers such as Convolutional and Max Pooling layers. These general modular components became a benchmark for most of the CNN structures used in modern computer vision tasks.

In general, there are 2 types of object detection CNN systems, namely 1-shot detection systems [45 - 47] and 2-shot detection systems [41 - 44]. Since 1-shot systems merged the region proposal system and object classification as one operation, their overall detection speed is often superior, with the cost of detection accuracy performance. Despite the difference in detection stages, most of CNN object detection techniques have a feature map extraction process typically at the start of the network system, serving as the backbone of the detection network [51, 52].

With these methods constructed, more researchers tend to utilize similar approaches for object detection when reconstructing 3D scenes. Some combined semantic cues and 1-shot networks to further improve detection [50], and some incorporated image depth map as input feature in the detection CNNs [48, 49]. Apart from the above approaches which used perspective images as inputs, object detection in panorama images have also been explored in various research. In particular, some researchers decided to split the panorama image input in multiple viewpoints, in order to still leverage the detection strength of the aforementioned popular object detection CNNs [19, 52]. Other methods still adopted panorama images as input but modified the network structure for the CNNs to adapt to 360 images as input, instead of perspective images [47,53,54,55].

Our approach to object detection is a mixed effort between [52] and [53]. We first split the panorama image into 4 parts with stereographic projection similar to [52]. Then, we send image partial inputs into M-RCNN, and use a Negative Feedback Mechanism similar to [52] to prevent duplicate detection. Predicted data such as mask, bounding box, and class will be sent to the CAD object retrieval stage. For the model, we take a M-RCNN system pretrained with MS-COCO dataset [57] and fine-tuned it with a mix of 2D-3D-S [26] and an extended SUN360 [25]. Experiments indicate the fine-tuned model can detect objects in panorama splits in stereographic projection.

Object retrieval is another topic of discussion in this chapter. It is the process of matching detected objects with its corresponding 3D CAD model in our CAD pool. We will cover the CAD collection in the next chapter. There are many methods so as to achieve object retrieval, such as training another CNN model [10, 58], and matching image with renderings of every CAD model from multiple viewpoints [59, 60]. We decided to use a similar approach as the latter and compute and iou between rendered CAD images and detected objects' ROI window. To ensure maximum accuracy, we extract the best 5 CAD proposals and ask the user to choose which best describes the ROI.

3.2 Related Works

Traditionally, many computer vision tasks, including object detection and image classifications, were mainly solved by feature descriptors such as SIFT [35], HOG [36], and SURF [56]. Although they have different working principles, they have the same function to extract low level features, for instance sharp edges and corners, from images. However, the features they detect are often hand-engineered, so they are non-capable of adapting detections to a specific set of data, and they operate the same way for every single image. Consequently, feature descriptors were often used concurrently with classical machine learning techniques, such as SVM [37] and kNN classifiers [38] to learn for data-specific tasks.

After AlexNet [40] showed how to operate image classification in large and complex ImageNet [61] using GPU's computation power and a deep layer structure, researchers started utilizing CNNs more frequently to tackle Computer Vision problems [39]. In terms of Object Detection, the problem can be divided into 3 distinct subtasks: Feature Extraction, Region Proposal, and Classification. The first subtask can be achieved through the use of feature extraction CNNs, such as V-9 GG [51] and ResNet [22], which are also known as the backbone of the detection algorithm. For the next two subtasks, some approaches consider them as separate operations and execute them consecutively. Examples include the R-CNN series, which include R-CNN [41], Fast R-CNN [42], Faster R-CNN [43], and Mask R-CNN [44]. They first propose regions where objects potentially exist, then classify each proposed region and output class confidence and regressed bounding box. In contrast to these two-shot approaches, one-shot approaches such as SSD [46] and YOLO [45] simultaneously detect regions and predict class score in a single operation, resulting a overall faster detection, but suffers from lower accuracy because the combined region proposal and classification forced these system to process larger amount of background than foreground samples [62].

Various previous 3D reconstruction research adopted the aforementioned CNNs to detect objects. Saliency map and YOLO detection results were combined to generate more accurate object predictions [50]. Another fused image depth map into the input

layer of R-CNN [49]. However, the aforementioned methods are only applicable with perspective images input, which cannot capture sufficient environment data to produce high quality virtual environment. Recently, research efforts are made such that Object Detection CNNs can operate recognition on 360 panorama images. Some decided to separate panorama images into parts and independently input them into detection networks [19, 52]. On the other hand, some approaches adjusted the network structure to adapt to panorama inputs. Extra multi-kernel network layer was added into Faster R-CNN to cope with panorama distortion [55]. Another approach used BlitzNet [47], a CNN based on SSD [46], and replaced all convolutional layers with EquiConvs [63], which has kernels specialized for equirectangular panoramas. Although directly detecting entire panorama images can be convenient, classic CNNs modified for panorama input can sometimes be less performant than the original version [63]. Therefore, we decided to separate panorama inputs into sections and leverage the unmodified version of Mask R-CNN for detection.

After object detection, object retrieval is required to obtain the corresponding CAD model for the detected object. To solve this problem, many approaches compare CAD renderings templates with the detected object image, with methods such as training a CNN model [10, 58] and HOG [60]. Some methods even entirely skipped model retrieval and adopt 3D cuboid boxes to represent objects in the virtual environment [17, 53]. Since we created our CAD model pool, we implement our own object retrieval system with inspiration from the template comparing approaches.

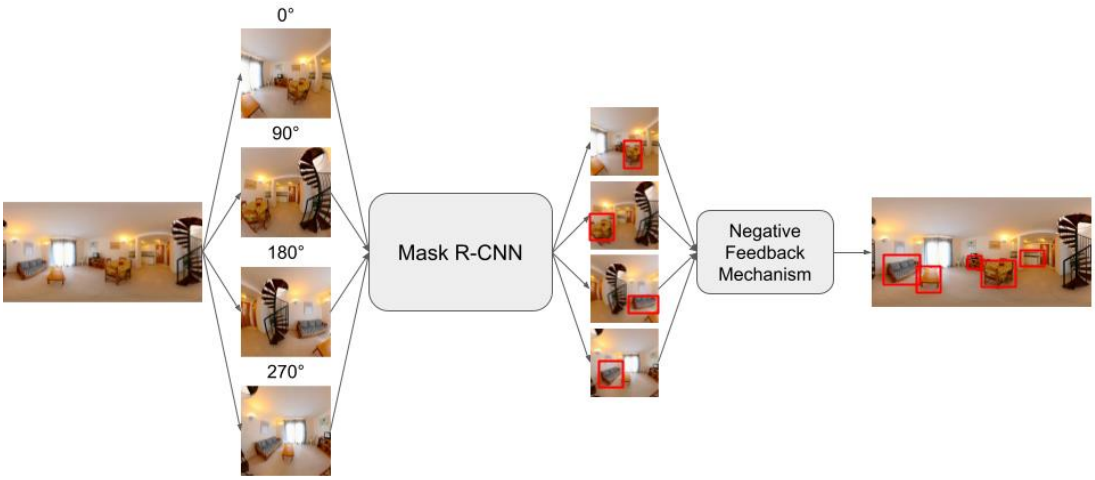

Fig. 4: Structure of our Object Detection Module

3.3 Methodology

We separate our object detection system into 3 main stages: Equirectangular Division and Reprojection, Mask R-CNN Prediction, and Negative Feedback Mechanism (Fig. 4). We will also explain our algorithm on Object Retrieval.

3.3.1 Equirectangular Splitting and Reprojection

The first stage of our object detection is Equirectangular Division and Reprojection. Inspired by [19] and [52], we first resize every input panorama into width 1024px, and height 512px. dimension. We then divide the equirectangular image into 4 distinct sections, each spanning 180° yaw and 180° pitch. We achieve this by rotating the original image horizontally by 0°, 90°, 180° and 270° respectively, and generate stereographic projections of the input image. The resulting reprojected sections are of 512x512px, capturing information of the indoor environment from 4 uniformly distributed angles. Note that each split overlaps the one next to it by 90°. Instead of perspective projection adopted in [19], we chose to use stereographic projection similar to [52], because it can preserve better visual information at larger FOV, with the expense of distorted straight lines [52].

3.3.2 Mask R-CNN Prediction

After the separation of the panorama image, we independently use them as input for object detection CNN. Here we chose Mask R-CNN [44] because its predicted mask can be used for object retrieval. As mentioned in the last section, Mask R-CNN is a two-shot detection system with a lower detection speed compared to one-shot detectors. We prioritize accuracy over speed because object detection is the most critical stage in the whole 3D reconstruction system. Its detection results define what will be seen in the generated virtual environment. Therefore, it is crucial to have as high detection accuracy as possible. For the structure of the Mask R-CNN, we base the network on a ResNet101 [22] backbone and used a mix of modified 2D-3D-S [26] and

extended SUN360 from [53] to fine-tune the model. More on the datasets used in the Experiments section.

After each divided panorama section has been inputted into Mask R-CNN, we obtain the ROI region in pixel coordinates, class, detection confidence score, and the inferred object binary mask for every detected object in the panorama split.

3.3.3 Negative Feedback Mechanism

Negative Feedback Mechanism was originally proposed by [19] to prevent misdetection and duplicate detections caused by the division of input panorama. Inspired by this idea, we implement our own Negative Feedback Mechanism. We first explain its importance, then elaborate on implementation.

The significance of our NFM is multifold. In general, object detection CNNs are relatively less performant when objects are occluded or truncated, specifically incorrect or missed detections may occur [50]. However, as our input is 360 panorama image, objects truncation is in theory non-existent. We make use of NFM to increase the probability where whole objects can be detected within the panorama images, unless occluded. Moreover, NFM can also minimize the distortion to detected objects due to stereographic projection. Finally, since 90° overlaps exist between the 4 panorama splits, the same objects are likely to be detected multiple times when they are being processed by Mask R-CNN. NFM recognizes same detected objects and therefore can solve the duplicate detection problem.

The workflow of the NFM is illustrated in Fig. 5. After we obtain the objects' data returned by Mask R-CNN, for every subdivided section from the panorama image, we locate the center of mass coordinates of the predicted object mask of every detected object. For example, if the network detects 4, 5, 6, 7 objects in the 4 reprojected image parts respectively, 22 sets of center pixel coordinates are calculated. We then recenter the stereographic projection to the center pixel coordinates for all detected objects. This effectively puts the object in question near the center of the projected split. We then execute object detection algorithm again with the recentered image, obtaining a more accurate prediction of the center object without occlusion issues. We repeat the

detection loop until all detected objects from the 4 panorama splits are recentered, after which we extract the recentered mask array and center yaw and pitch angles. It is worth noting that [19] only operate NFM on objects that are originally detected at the edge of panorama splits, while our approach runs NFM on every detected object. By moving objects to the center of the projection, we minimize the distortion on the objects' ROI, because stereographic projection visual distortion occurs the least at the image's center.

After the recenter process, we combined prediction data across all panorama splits to form the final detections. To prevent potential duplicate object detections, we cross referenced every detection to each other, and grouped similar ones based on class and center angles. As panorama splits overlap each other at strictly 90°, each detected object must appear in 2 or more splits. Therefore, we consider one-time detections as misdetections, and prune them from the final detected candidates. Fig. 6 shows final detection results with their respective panorama input.

Fig. 5: An example of the recentering process during NFM

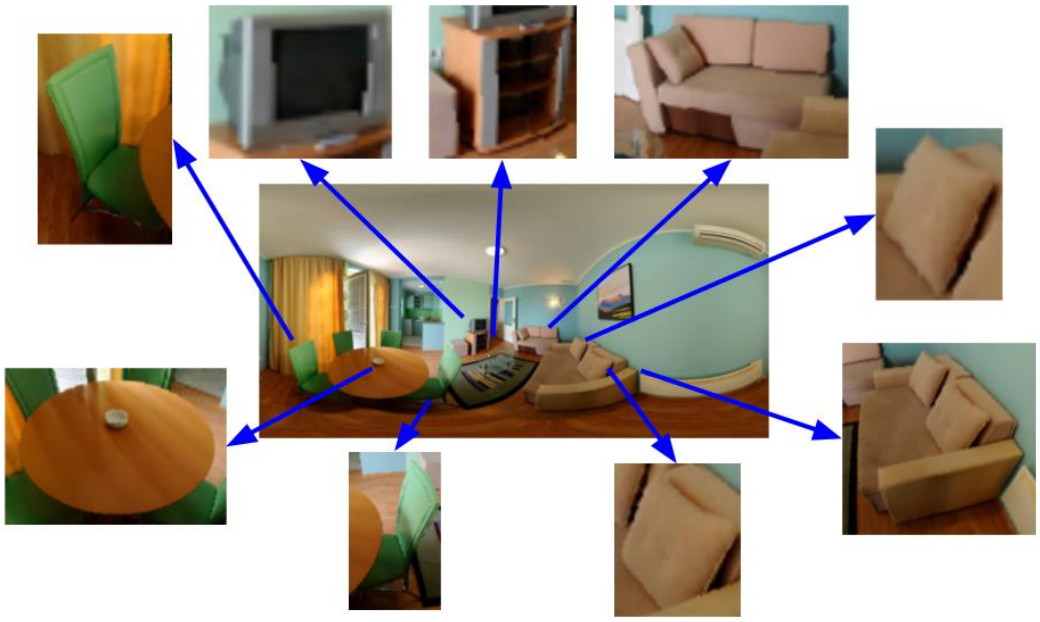

Fig. 6: A panorama image input and its prediction results of our objection detection module

3.3.4 Object Retrieval

3.3.4.1 General Workflow

Our approach on CAD model retrieval is partly inspired by ObjectNet3D [59] and IM2CAD [10]. Similar to their approaches, we also match detected objects' ROI with rendered images of CAD models.

We first utilized Blender [64] to render images of all CAD models within our CAD model pool across multiple viewpoints. Specifically, we set up virtual cameras around a radius 5m sphere and point cameras to the center, where the scale-normalized CAD models are. We sample rendered images with a 5° azimuth angle, and 30° elevation intervals. Also, we only sample images where the virtual camera is located at the upper hemisphere, as objects in our categories, which most of them are furniture, rarely have rotational pose that possess negative elevation. This results in a total of 216 rendered viewpoints for each CAD model.

Next, we determine the degree of overlapping between rendered images and the detected ROIs from our fine-tuned Mask R-CNN. To eliminate the factor of RGB

information, we extract binary masks from the two and calculate their IoU scores. We then use the viewpoint with the max IoU score to represent the specific CAD model. The same ranking operation is executed for every CAD model with the same category as the detected object. Finally, to ensure accuracy, 5 CAD models with the highest score are voted out, and we let the user decide which of the 5 best represents the detected object.

3.3.4.2 Solution to Occlusion

If detected objects are occluded by others, it is impossible for Mask R-CNN to correctly estimate its object mask. This creates inconsistency between the predicted mask and CAD model's rendered images, which can severely harm IoU accuracy. We propose an enhancement algorithm to alleviate this problem.

For each detected ROI, we assume the bottom of the object is always visible and not occluded. During the matching of rendered masks and detected ROIs, we rescale the ROI so that it fits the rendered image without any extrusion. Since the ROI's dimension is usually not identical to the rendered mask, either the resized ROI is "thinner" than the rendered image (same height, smaller width), or it is "shorter" (same width, smaller height). For the latter case, we snap the ROI to the bottom of the rendered mask and calculate IoU. For the former case, we measure IoU score 3 times, each snapping the ROI to the left, right, and center of the rendered mask before calculation. Fig. 7 illustrates an example of the algorithm in action. By trying to overlap masks in different locations, we can estimate the best IoU score even when an arbitrary part of the detected object is occluded.

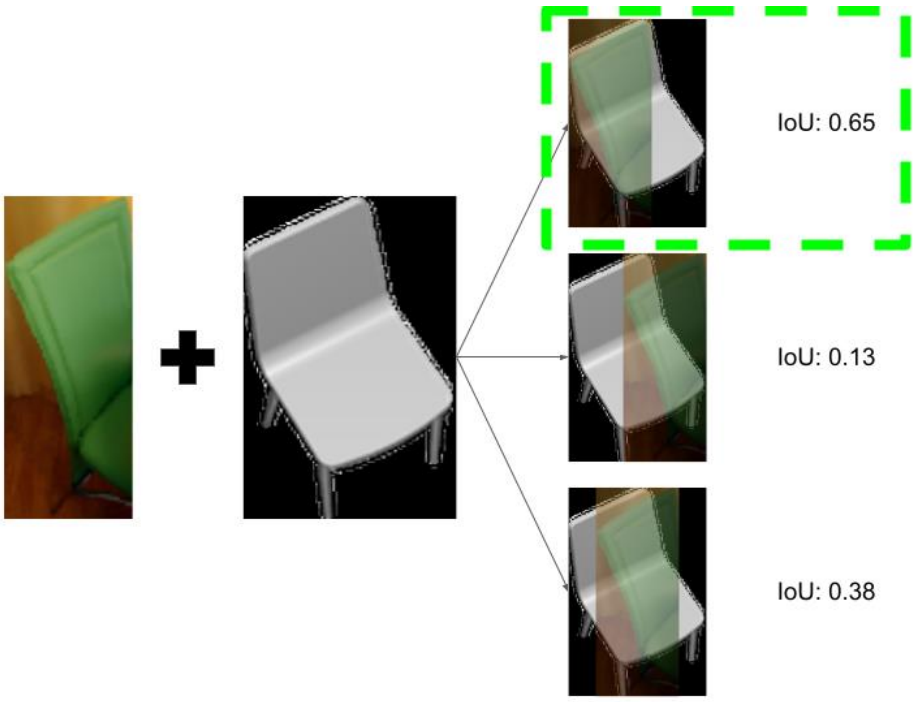

Fig. 7: Example of our Object Retrieval system's solution to occlusion

3.4 Experiments

3.4.1 Datasets

For datasets, we use a mix of Extended SUN360 from [53] and 2D-3D-S [26] for training. For Extended Sun360 (referred as SUN360Ext below), there were originally 566 train images and 100 test images, all in equirectangular panorama format. We notice that [53] extended the dataset by annotating object masks onto every object in the panoramas. Fig. 8 illustrates an example of the new annotations. However, since [53] predicts all objects as cuboids, its ground truth segmentation shapes are confined to be cuboid as well. As a result, objects that possess unique contours, such as chairs, are represented incorrectly and unrealistically. Therefore, we reannotated all SUN360Ext's masks again, but closely following every objects' natural shape. Fig. 8 represents one example of our mask annotations. Furthermore, since we decided to put reprojected sections of panoramas into Mask R-CNN for object detection, we divided every panorama in 4 splits with the same setup introduced in the Methodology section.

Each split is of stereographic projection, so unlike conventional perspective images input, straight lines are not preserved. In total, we have 2258 train images and 399 test images.

For 2D-3D-S dataset, there were originally 732 panorama images. We again look at the annotated mask and noticed a massive downgrade from SUN360Ext. Compared to SUN360Ext, object masks follow object contours, but the quality is extremely coarse, and masks are incomplete sometimes. Fig. 9 shows an example for this. Notice that black gaps exist within the red bookcase mask, as well as the blue chair below it. Annotations with low quality seriously impact the training performance of CNNs, so we again reannotated some masks. Similar to SUN360Ext, we also separated the dataset's panorama into 4 sections each. In the end, we extracted 1117 train images and 198 test images in this dataset. We originally set ratio of the two datasets to be 50-50, but we worry that some low-quality mask annotations will harm inference accuracy, so we set the ratio of SUN360Ext : 2D-3D-S to around 7 : 3.

3.4.2 Training Network Parameters

We fine-tuned a Mask R-CNN with ResNet-101 as backbone. We initialize the training process with a network pre-trained with the COCO dataset [57]. COCO dataset contains similar classes as our own 10 categories, so it serves as a good starting point for the network. We keep anchor settings as default and set the batch size to 8. We also limit each input image dimension to be 512x512px. We train our network with 4 NVIDIA Tesla V100 DGXS for a total of 320 epochs, with a learning rate of 0.001 and momentum of 0.9.

3.4.3 Detections and NFM

After the Mask R-CNN is finetuned with our modified datasets, we prepare inferred data for evaluation. Specifically, we execute the inference process on all images in SUN360Ext test set. It is important to note that instead of merely inputting split panorama sections and outputting predictions on individual splits, we input the whole panorama and ran the whole object detection module from panorama division, and

reprojection, to processing detected objects from all splits and utilizing the NFM for deduplication. In short, we take all 100 test set panoramas as input and output objects detected in the 360 panoramas.

3.4.4 Object Retrieval

To better describe the results of our Object Retrieval algorithm, we first run Mask R-CNN object detection on all SUN360Ext test set images. Then for every detected object, we run the object retrieval algorithm and match objects against our 3D CAD object pool. We obtain 5 best matches for each detection and save it for evaluation in the next section.

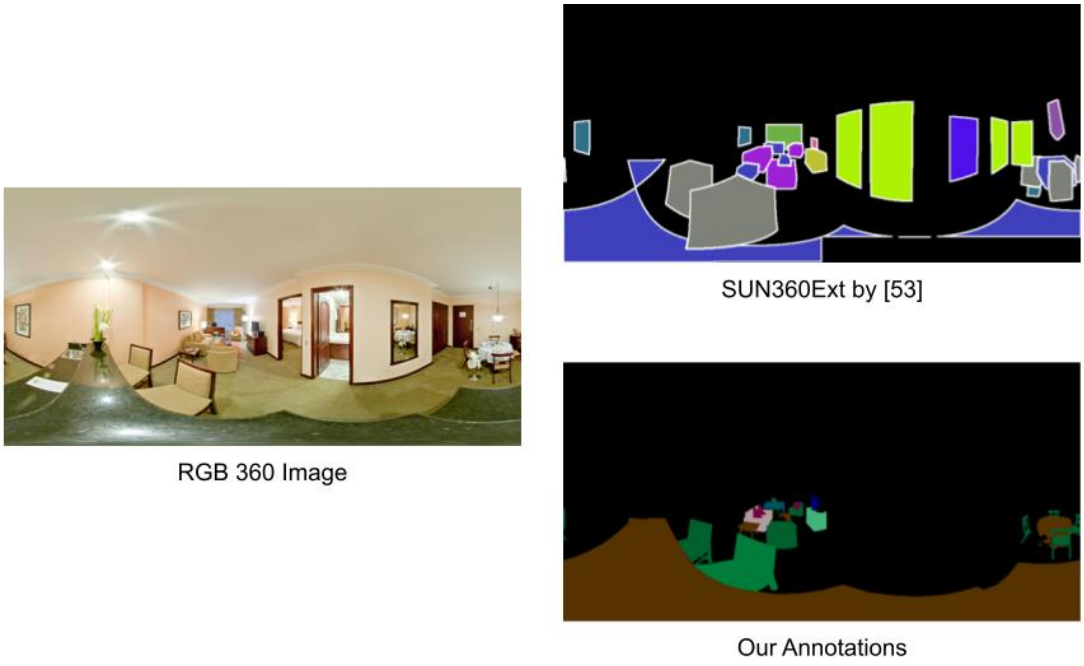

Fig. 8: Comparison between [53]'s and our semantic segmentations

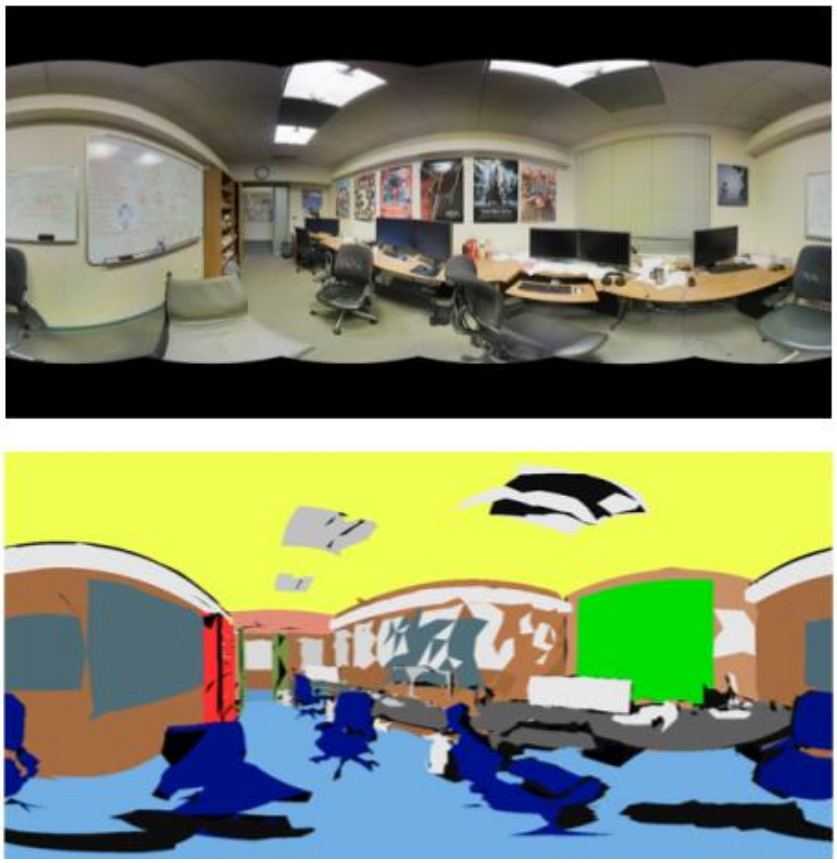

Fig 9: Example of semantic segmentation of 2D-3D-S. Notice masks are quite incomplete and coarse

3.5 Evaluation and Results

3.5.1 Evaluation Methods

3.5.1.1 Mask R-CNN and NFM

To test the robustness of our object detection module and evaluate detection results against ground truth, we use the popular object detection metric mean Average Precision (mAP). The mAP is defined with two significant metrics, precision and recall rate. Precision determines how accurate the model is concerning all its positive results, and recall determines how well the model can actually detect real positive samples, as opposed to how often the model misses objects that it is supposed to detect (False Negative).

To define a successful detection, we use IoU as a standard metric. If the prediction mask is overlapping with ground truth mask labels for over 0.5, the detection is regarded to be True Positive, else False Positive. Although the typical threshold of positive detection is 0.5, we notice that objects' scale difference within panorama images are rather large. Some objects closer to the camera appear large, while objects closer to the walls appear small, depending on the size of the actual indoor environment. In the latter case, the effect of mask pixel difference against the IoU is significantly greater than that of larger objects, because small object masks do not have many pixels anyway, which restrict the room for error. To counter this problem, we set another IoU threshold, 0.2, to relax the mask overlap condition needed to be regarded as successful detection. Finally, the AP is calculated by recording the area under curve (AUC) of the precision-recall curve, and mAP is computed by averaging scores across all categories.

3.5.1.2 Object Retrieval

We demonstrate qualitative retrieval results by first coupling the best 5 CAD matches with the detected ROI. We then manually choose significant examples to address, such as both successful and failed cases, and discuss the reason behind the results. We try to uniformly select examples from every category for fairness.

3.6 Results

3.6.1 Object Detection Module mAP

	Beds	Bookcases	Bottles	Chairs	Desks	Pillows	Sofas	Tables	TV	Wardrobes
AP @ 0.2	0.8478	0.3043	0.053	0.4483	0.3072	0.4612	0.6431	0.4682	0.6991	0.4527
AP @ 0.5	0.7667	0.1177	0	0.2861	0.2483	0.3851	0.6213	0.3403	0.6991	0.4190

mAP @ 0.2	0.4685
mAP @ 0.5	0.3884

Table 1: mAP and AP results of our fine-tuned Mask R-CNN model

Table 1 shows the Average-Precision score and the mean AP across all categories. Inspired by [52], there are also AP @ 0.2 and AP @ 0.5, which means AP with IoU threshold 0.2 and 0.5 respectively. As expected, AP @ 0.2 overall has a higher score than AP @ 0.5, because it is easier to have a smaller overlap between prediction and ground truth masks. mAP @ 0.2 is 0.46845, while mAP @ 0.5 is 0.38837. Compared with the object detection system in [52], our system's detection result is actually competitive with theirs, which fluctuates between 0.30 to 0.43 under various conditions.

Next, we will analyze the AP score within some of the classes. We notice that, in general, objects that are usually large-scale in panorama images have a higher AP score. For example, Sofas and Beds have two of the highest AP @ 0.5. We believe this is because larger objects are usually easier to be detected in the first place, which increases the recall rate. Also, larger masks allow higher tolerance for pixel error, and therefore it is easier to obtain higher IoU, hence higher precision. We then look at some extreme cases. First, we can see that TvMonitors, despite being rather small sized, have a high AP of 0.7, and with the same score in both IoU thresholds. We think the high score is due to the consistent shape and appearance of TvMonitors in most images. For other categories, when the object is rotated in different orientations, the object changes a lot in appearance, so a larger number of images data is needed to fully learn the visual properties of some objects, take chairs for instance. However, TvMonitors are always black in color and their appearance does not alter much in images (a black square), therefore it is more straightforward for Mask R-CNN to learn what TvMonitor is, resulting in a higher AP. Another extreme point is Bottles, which is relatively low in AP score. We suspect this is because of the lack of data in the dataset. In the SUN360Ext test set, there are only 19 bottles in all of the 100 images, and its rarity

makes it harder to make accurate predictions. Moreover, bottles are small compared to other furniture, which makes them even naturally tougher to detect as well.

3.6.2 Qualitative Analysis: Object Retrieval

Fig. 10: Examples of retrieved object, including their original corresponding ROI

Fig. 10 shows some qualitative examples for CAD models retrieved from our Object Retrieval algorithm. The leftmost image is the detected object's ROI, and the 5 images at the right are the best 5 CAD hypotheses generated by the algorithm. The one with the highest IoU score is the closest to the detected object and decreases to the right. We will explain both the advantages and disadvantages of the algorithm's current state.

One of the advantages of the current algorithm, is that the IoU metrics can effectively give a coarse context about the scale of the detected object. For example, in the last

wardrobe object, its width is longer than the height, making it appear “flat” as opposed to “tall”. The object mask can capture this information and match it with CADs that are with similar dimension ratio. Consequently, we can see the first 3 proposed CADs have the same “flat” scale as the detected object.

Another advantage of the current system is that it can deal with occlusion to an extent. Notice the first chair object is occluded by the table on the right. Even then, our algorithm can still output similar CADs, and even make a fair coarse estimation of the 3DOF pose.

However, the biggest disadvantage of the current approach is its inability to capture CAD/Object interior features. For instance, we look at the 3rd set of images of the sofa. If only the object mask is extracted, we have no information about if the sofa has side handles or not. As a result, the best CAD match is a sofa without handles. As we can see, the current algorithm cannot allow us to leverage individual CAD objects’ interior features to aid in selection. It can only match objects to CAD based on the general shape of the object.

Chapter 4: Object Pose Estimation and Pose Estimation Dataset

4.1 Introduction

Object pose can be divided into two main types: 3DOF pose, and 6DOF pose. 3DOF pose refers to the spatial rotation angles of an object, while 6DOF pose implies both spatial 3D location and orientation data of an object.

Compared to object detection, 3D pose estimation is a relatively new topic, as active related research only started around a decade ago. Previous approaches to the problem can be concluded into 2 big categories: Methods that utilize data from 3D models for estimation, and methods that do not.

For the former type of approach, various methods have been proposed to harness data from 3D shapes to estimate object pose. One of the popular ways is template-matching, where researchers made use of multiple viewpoints of 3D models to match with the object image, similar to our approach to Object Retrieval [65, 66, 67]. Feature-based methods are another common approach for leveraging 3D shapes in pose estimation. In this methods, features from the object image, such as semantic keypoints and 3D bounding box projected coordinates, are first predicted, then using 2D to 3D conversion algorithms [68], compute the corresponding 3D pose of objects [69 - 73].

As 3D shape data is not often available, and 3D models are not always completely visually identical to the observed images, researchers also looked into methods that do not require 3D shape information. In general, these methods treat pose estimation problem as a classification or regression problem, and directly estimate 3DOF or 6DOF pose through CNNs [74 - 77].

Our approach utilizes PoseFromShape [78] as our object viewpoint estimator. It modeled its implementation using both categories of method, leveraging rendering templates from 3D shapes. as well as considering 3DOF pose estimation as a mixed

problem of classification and regression. In practice, when an image and a 3D CAD model are provided to the system, PoseFromShape will output the corresponding azimuth, elevation, and in-plane rotation angles of the observed object. Experiments show PoseFromShape can produce satisfactory rotation pose estimation from the ROI detected from our Object Detection System.

Aside from the 3D reconstruction system, we also propose an Object Pose Estimation dataset specified for indoor environment, and a Unity-based annotation tool coupled with the dataset. 3D Pose Estimation datasets can be again categorized in 6DOF datasets [65, 76, 79], and 3DOF datasets [59, 80, 81. 82]. 6DOF data generally include images of different objects, their 3D shapes, and are annotated with the objects' location, rotation and their 3D bounding boxes. On the other hand, 3DOF data also mainly feature images and 3D models, but are not annotated with 3D translational data. Typically, current 6DOF datasets only focus on small scaled objects, such as driller, iron, and lamp [65]. However, 3DOF datasets are capable of including objects from all scales, from bottles to boats [59]. Most of the datasets offer an annotation tool for users to annotate data for the dataset. Depending on what kinds of data is included in the annotations, annotation tools differ in complexity in terms of functions. For example, Pascal3D+ [80] and Pix3D [81] have implemented a keypoint feature to let annotators label keypoints on 2D images. However, ObjectNet3D [65] does not include this type of annotation.

Our approach to the pose estimation dataset, is to make overall improvement to the existing 3DOF datasets by combining their features and functions together. Apart from the keypoint labeling feature mentioned above, we allow annotators to adjust the camera's extrinsic parameters similar to ObjectNet3D and Pascal3D+. This feature is nonexistent in Pix3D, so we include it to increase our dataset's robustness. Besides, we quantitatively experimented on the quality of our hand-made annotations. Results show that we have overall higher 2D-3D alignment annotation quality than some existing datasets.

4.2 Related Works

This section covers previous relevant approaches regarding pose estimation and pose estimation datasets.

Object pose estimation approaches can be organized into two main types: Pose estimation with 3D geometry information, and Pose estimation without 3D geometry information.

The former can be again divided into two subtypes, namely template-matching approach and feature-matching approach. Template-matching approaches compares object in the input image with a fixed number of viewpoints of 3D model, and the most similar viewpoint will be regarded as the 3D pose [65, 66, 67]. For example, [66] generated viewpoint templates for every available 3D model and trained a custom CNN descriptor, such that the Euclidean distance between descriptors of the input image and that of the best-pose rendered image, will be the smallest. However, these matching algorithm alone is prone to error if the object in the input image is truncated or occluded [78]. Feature-matching methods, on the other hand, predict visual feature from input image and estimate 3D pose by corresponding detected features with predefined features of the 3D models. For example, a popular approach is to estimate keypoints or 3D control points projections in the input image and adopt PnP algorithm [68] to convert 2D projections into 3D space, consequently computing the corresponding pose of the 3D model [69, 70, 71]. The limitation of these approaches, however, is the difficulty to estimate keypoints if the observed object is textureless [78].

Recent research tends to focus on pose estimation methods that do not require 3D geometry information, as 3D models' shape may not be consistent with the input image itself. In general, these approaches regard pose estimation as output of CNN networks and apply classification or regression to learn output pose values For example, [75] classify object 3DOF viewpoint directly by training a class-dependent CNN with synthetic images with artificial clutters. [77] first used a DRN [83] to perform semantic segmentation, and trained a pose interpreter network, consisting a resnet-18 [22] followed by an MLP, to regress object pose by taking detected object masks as input.

Among the two styles of approach, PoseFromShape [78] adopted both styles in its approach by using template viewpoints or point cloud as input parameter, and predict object pose by classification and regression with ResNet-18 [22] and MLP. We decided to apply this approach because of the ease of deployment in our system, and its state-of-the-art estimation performance.

As for pose estimation dataset, we can summarize object pose datasets into 3DOF and 6DOF. Famous 6DOF pose datasets include LINEMOD [65], and Occlusion [79], which is a subset of LINEMOD. LINEMOD is a dataset of videos over 1100+ frames, containing 15 objects in the videos. Frames in the videos capture a clutter of objects in a confined space, from different viewpoints. The frame images are annotated with 6DOF pose of the objects, as well as the objects' class and 3D mesh. However, the 15 object categories are all smaller than a table, which are not common in indoor environments such as living room and bedroom, where medium to large scale furniture are dominant. On the contrary, 3DOF datasets, such as Pascal3D+ [80] and PIx3D [81], can contain image-3D shape pairs of larger objects such as tables and chairs. These datasets are relatively relevant to our research for 3D reconstruction. We therefore base our own dataset on 3DOF datasets.

4.3 Methodology

In this section, we will explain our approach on estimation object pose estimation, elaborate on the construction of our new pose estimation dataset, and its accompanying annotation tool.

4.3.1 Object Pose Estimation

We estimate object's 3DOF pose using PoseFromShape [78]. Specifically, we take the ROI detected from our object detection system, and the retrieved 3D CAD model from the model retrieval algorithm, and input into the PoseFromShape neural network system. The 3D model is first preprocessed by generating rendered images from different viewpoints. Here we customise the viewpoint rendering process to be identical to our object retrieval system. It samples from the same angles, with a total of

216 rendered images, and only samples from the upper hemisphere around the object. Therefore, we can reuse the rendered images for pose estimation as well to optimize space complexity. PoseFromShape then concurrently extracts features from input images using ResNet-18 [22], and from the rendered images using another CNN. After that, feature vectors are concatenated and put into an MLP, where azimuth, elevation, and in-plane rotation angles are estimated. To be precise, possible 3DOF poses are separated into a finite number of bins, then the estimation process classifies object pose into one of the bins as a coarse prediction, and outputs an offset angle for regression and finer final estimation. PoseFromShape outputs the inferred object Euler angles, and we use them for the final reconstruction phase of our system.

4.3.2 Pose Estimation Dataset and Unity-based Annotation Tool

Our goal for this new dataset is to create a pose estimation dataset specific for indoor reconstruction, and make overall improvements compared to the existing 3DOF pose datasets [59, 80, 81, 82], by combining their advantageous features into a single dataset. We aim to make enhancements based on: 2D-3D alignment quality, 3D model diversity, Image source diversity, Number of annotation function and annotation flexibility. In this subsection, we explain how we collect our data and explain our enhancements regarding our dataset. Fig. 12 - 14 shows statistics for our dataset.

4.3.2.1 Categories

As our dataset put emphasis on indoor room reconstruction, the categories in the dataset are all commonly-seen objects in living rooms, bedrooms, or offices. There are a total of 10 distinct categories: Beds, Bookcases, Bottles, Chairs, Desks, Pillows, Sofas, Tables, TvMonitors, and Wardrobes.

4.3.2.2 Images

We collect images from the 10 categories from several sources. Similar to Pascal3D+ [80] and ObjectNet3D [59], we chose the popular image classification dataset ImageNet [61] as the main source. To add diversity to the image source, we also crawl object images from Google, Bing and Baidu if images from ImageNet are insufficient.

We search images using keywords the same as category names. Compared with existing datasets, we have an overall higher diversity of image source, supporting more possible poses in the dataset (Table 2). To ensure image quality, we manually filter images that are overly truncated or occluded, as well as those that are taken outdoors. We believe the data cleaning step is one of the crucial steps, because the viewpoint of some images from ImageNet are extremely close to the object, which caused the object to be truncated at all 4 sides of the images Fig. 15. We consider these images unsatisfactory, because although pose estimation is somewhat possible, they rarely appear in panorama images where the camera is usually located at the center of the room.

In total, we collected 19543 images from the 10 specified categories. Fig. 16 shows some examples of the images in our dataset.

4.3.2.3 3D CAD Models

We collected a pool of 3D CAD models from Trimble 3D Warehouse [84], which is a public 3D shape repository used by every existing 3DOF datasets mentioned above. We downloaded all 3D models in .dae format, and imported them into Unity for preprocessing. For every model, we normalized their scale to a unit sphere and reorientated them, so that their forward axis is -Z, and upward axis is Y axis in Unity. We also dispatched specific parts of some 3D models to better fit their respective categories. For instance, since we regard pillows and sofas as two different categories, we delete every pillow on top of sofas if they are modeled as parts of them. We also delete curtains in beds because we believe they are irrelevant to the 3D models. Finally, we manually added keypoints to individual 3D models. Relative keypoint locations within the same class are similar. For example, keypoints in chairs are always anchored at 4 corners of the seat, the bottom tip of 4 legs, and at both ears. On average, we set 11 keypoints for every 3D shape. The function of the keypoints will be explained in the next section. Fig. 17 shows some examples of the 3D models with keypoints. Furthermore, we show some statistics regarding 3D models among the aforementioned 3DOF datasets. Currently, we have 198 objects in our 3D CAD pool. Although we have less number of models compared to PIx3D, which has 395 models, the number can still

compete with IKEA dataset [82], and is significantly more than Pascal3D+. Comparing with ObjectNet3D, despite that fact that it has the largest 3D shape scale obtained through ShapeNet [85] and 3D Warehouse, it only has 783 models with keypoints. Since it has 100 categories, this implies ObjectNet3D has merely about 8 models with keypoints per category, while our dataset has, on average, 20. We believe having more 3D shapes can diversify the model choices for each image, and it will be easier for the system to retrieve the best-suited 3D model.

4.3.2.4 Annotation Process

We explain our Annotation Tool and its functions in this subsection.

After we obtained both images and CAD data, we constructed an annotation tool in the Unity engine to ease the annotation process. In the application, we set up a GUI to display the 2D image, and a virtual camera display pointing at the selected 3D model. For its extrinsic parameters, we set the focal length to 1, x/y sensor size 0.5, camera resolution to 500px by 500px, and aspect ratio to 1. The annotation tool offers the following annotation features:

Bounding Box Labeling: The object bounding box can be specified by enclosing the object with a resizable UI box. This feature was implemented in ObjectNet3D, but not in Pix3D and Ikea. For Pascal3D+, it can load bounding boxes if the data is provided, but it cannot be adjusted (read-only).

2D-3D Alignment: Annotators can browse through the CAD pool and select the best corresponding 3D model. The selected model can be aligned with the object displayed in the image by adjusting the camera principal point, object Euler angles, and distance between camera and object. This flexibility of parameter adjustment is unique to Pascal3D+ and ObjectNet3D.

Keypoint Labeling: After confirming the 3D pose, annotators can manually indicate the 2D pixel coordinates of corresponding keypoints on the 3D CAD model. Annotators can also specify if the current keypoint is occluded, truncated, or invisible (self-occluded). This feature is only not available in ObjectNet3D.

Table 3 shows a clear comparison between annotation tool functions of different datasets. Our dataset is the only dataset with all the features implemented. Object bounding box labeling allows the dataset to be used for joint object detection and pose estimation; Manual pose parameter adjustments can maximise annotation flexibility; Keypoint labeling function grant users with the ability to train keypoint-feature-extraction based CNN to estimate 3DOF pose outputs.

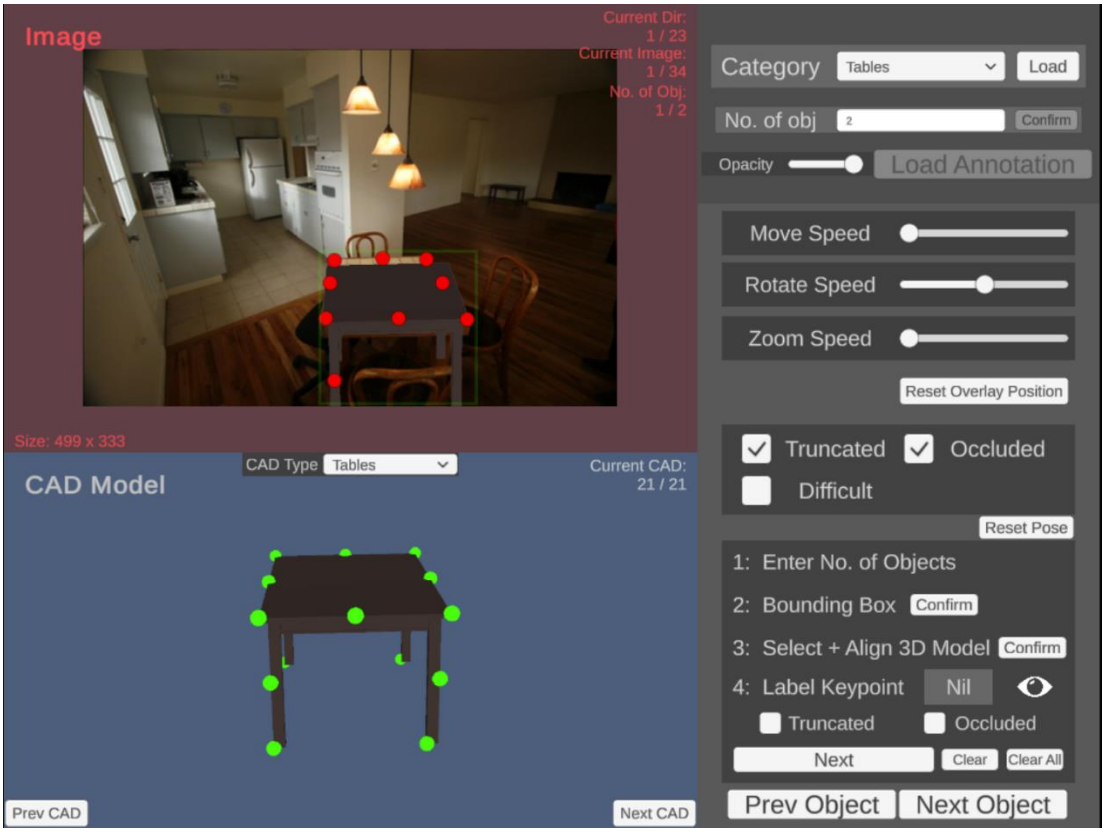

Fig. 11 A screenshot of the annotation tool.

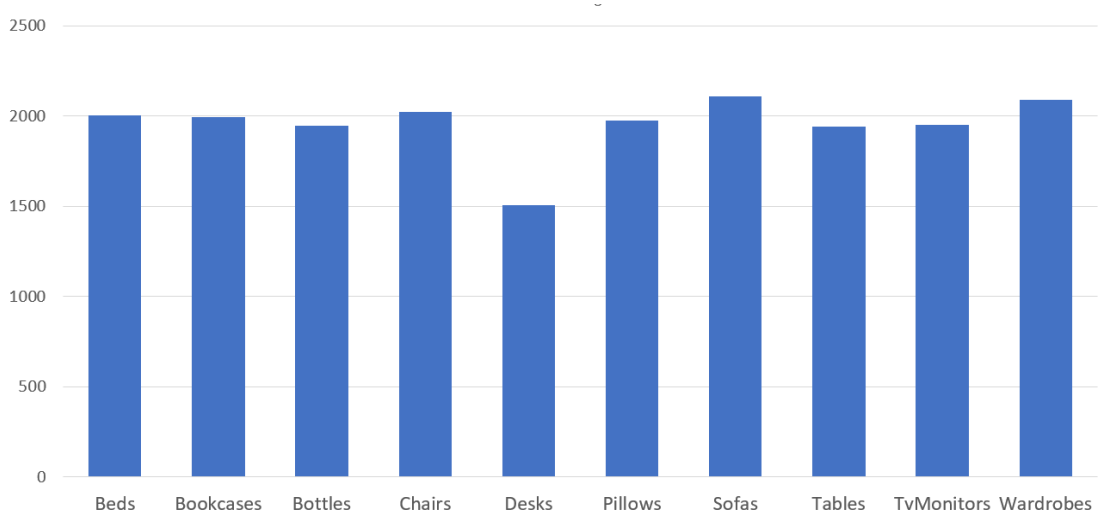

Fig. 12: Number of images per category

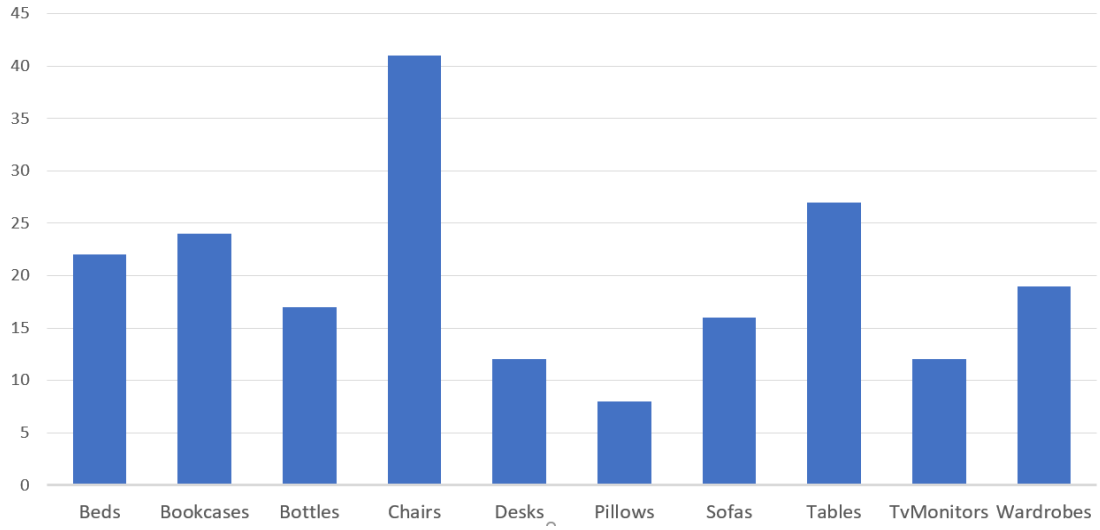

Fig. 13: Number of CAD model per category

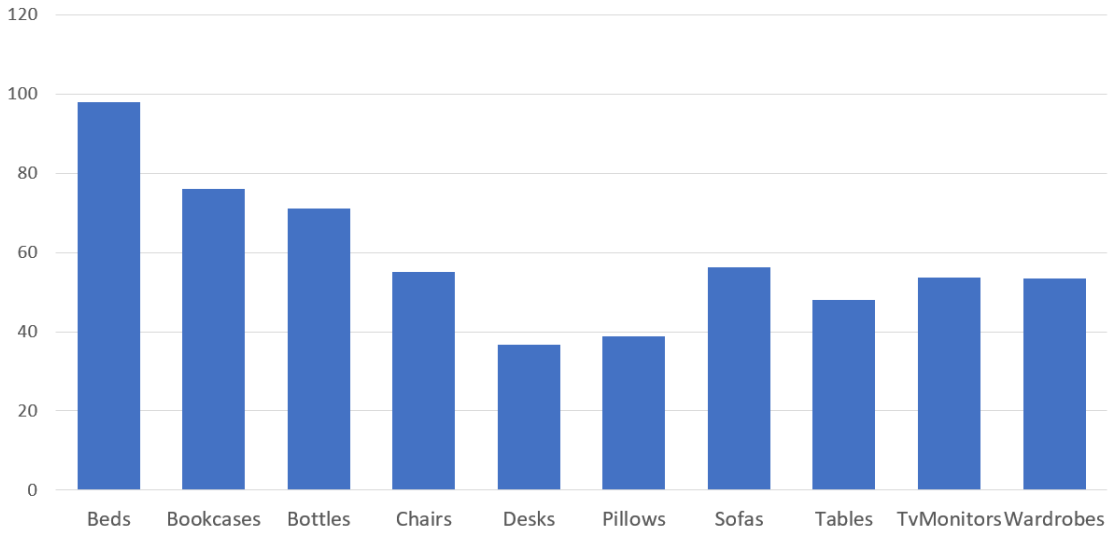

Fig. 14: Average number of images per CAD model

	PASCAL VOC	ImageNet	Google	Bing	Baidu	Flickr	No, Of Images
Pascal3D+	√	√					30899
ObjectNet3D		√	√				90127
Pix3D			√	√	√	√	14600
IKEA						√	759
Ours		√	√	√	√		19543

Table 2: Comparison between image source and image number from existing datasets

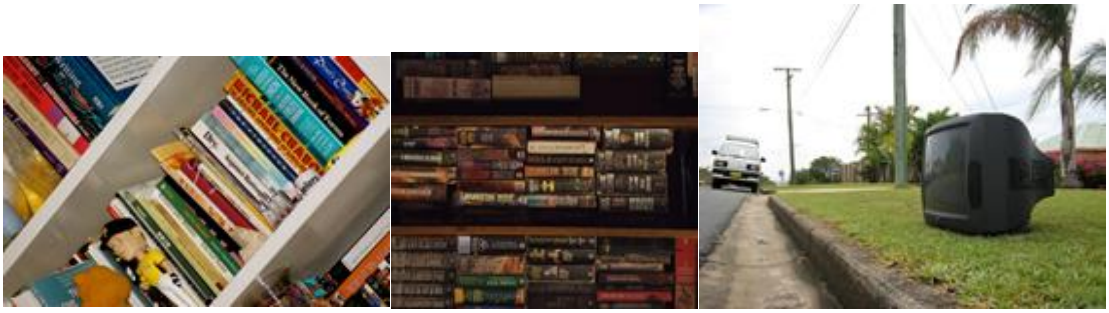

Fig. 15: Examples of images from ObjectNet3D (ImageNet) that require pruning for indoor object pose estimation task.

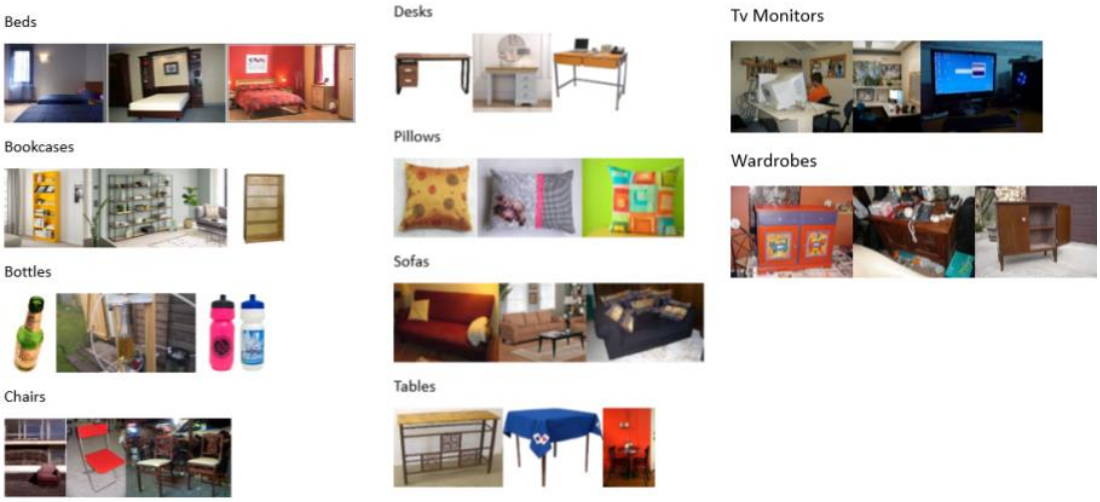

Fig. 16: Some examples of images in the dataset

Fig. 17: Some examples of CAD models in the dataset

	Pascal3D +	ObjectNet3D	Pix3D	IKEA	Ours
Bounding Box Labeling		√			√
Adjustable Camera Parameter	√	√			√
Keypoint Labeling	√		√	√	√

Table 3: Comparison between annotation features across all existing datasets

4.4 Experiments

4.4.1 Pose Estimation

In order to assess the effectiveness of our Pose Estimation Module, we design the following experiment to compare the 3DOF pose between PoseFromShape predictions and ground truth pose from our pose estimation dataset.

In our pose estimation dataset, we first randomly selected 10 images from each 10 categories, resulting in a total of 100 images. We then traced the image's annotation file and extracted the ground truth 3D CAD model used to align with the image's observed object. After this step, we obtained 100 image-CAD pairs for inputs of pose estimation.

Before inserting the image-model pair into PoseFromShape, we initialized a PoseFromShape model that is pre-trained with the ObjectNet3D dataset. Although PoseFromShape offered a few pre-trained models to users, we chose ObjectNet3D as pre-trained weights because the ObjectNet3D dataset has the largest scale out of the existing 3D pose datasets, having images of 100 categories. Most of our own 10 categories can be found inside the ObjectNet3D dataset. Therefore, we believe using this pre-trained model is the most compatible.

After setting up the pose estimation model, we input the 100 image-CAD pairs into the module as input. More specifically, since PoseFromShape utilizes a template-based approach to extract CAD model features, we actually input multiple viewpoints of the target CAD model rendered in Blender, instead of the .obj model itself. To optimize runtime of the actual estimation, the viewpoint renderings of all 198 CAD models are generated before the estimation process and are saved for future retrievals. As mentioned in the above section, each CAD has 216 viewpoints assuming only the upper viewing hemisphere is considered.

Upon completion of pose estimation of all 100 object images, we obtain a set of {azimuth, elevation, in-plane} angles as predictions of PoseFromShape. We separately extracted the corresponding ground truth angles from the annotation files of the same 100 objects and saved them together as pairs for comparison.

4.4.2 Pose Estimation Dataset

We conduct another experiment similar to Pix3D [80] in order to evaluate the quality of annotation, specifically the 2D-3D alignment, in our new dataset.

The first part of the experiment is actually quite similar to the previous one. We again randomly select 100 trial images from our custom-made pose estimation dataset, 10 from each category. Note that these 100 images are not the same as those from the previous experiment. We then manually added segmentation masks to every one of these objects. After that, we extract the annotated {azimuth, elevation, in-plane} rotations from all objects, as well as their corresponding CAD model, and used Blender [63] API to dynamically generate an image of the CAD model, with the annotated

3DOF pose. We create a mask for the rendered image afterwards. Essentially, what we did for each of the 100 test images, is that we generated a man-made object mask, as well as a CAD mask, which comes from the 2D-3D alignment between image and CAD model during the annotation phase of this dataset. Ideally, two masks would be the same if the CAD model is completely identical to the object in the image, and if the annotated 3DOF pose is the same as the image. The more similar the two masks, the higher the annotation quality of our dataset.

4.5 Evaluation and Results

4.5.1 Evaluation Methods

4.5.1.1 Pose Estimation

After the experiment, we obtained 100 sets of PoseFromShape prediction 3DOF angles, as well as their corresponding ground truth angles. In order to estimate the accuracy of pose prediction, [19] computed the angular error between predicted angles and ground truth angles. Inspired by this method, we adopted a similar metric to evaluate our pose estimation module. To be precise, we first converted both Euler angles into quaternion, and computed the absolute distance between the pair of orientation values. All quaternions are normalized to unit quaternions of magnitude 1, and the maximum absolute distance between two quaternions is $\sqrt{2}$. The lower the absolute distance, the more similar between the 2 3DOF pose in question. For all categories, we calculate and sum up the quaternion distance for all 10 selected objects and compute a mean angular distance for each category for evaluation. It is worth noting that, at this current stage, we did not fine-tune PoseFromShape using our dataset. Therefore, we expect the average error might be larger compared to existing approaches.

4.5.1.2 Pose Estimation Dataset

We obtained two masks per test image after the above experiment. One is a manually annotated object mask, another is produced by applying the annotated 3D pose of the

object onto it and projecting the model onto the 2D image plane. We decided to measure their similarity with the IoU between 2 masks.

Using IoU as the metric here has two significances. First, it indicates that the 3D CAD model is selected sensibly. Every CAD object, no matter inter-category or intra-category, has a different scale ratio. For instance, some bookcases have wide but short dimensions, while some have tall but thin dimensions. Selecting the suitable CAD model during annotation can ensure no parts of the image or 3D model is "sticking out" due to scale ratio difference, hence increasing IoU. A high IoU can also imply the 3D pose is well annotated, because the more accurate the annotated angle, the more is aligned between 2D and 3D, which ultimately leads to an increase of mask overlap.

We separately sum up the IoU score from different categories and compute the average overlap score within each category.

4.6 Results

4.6.1 Pose Estimation

	Beds	Bookcases	Bottles	Chairs	Desks	Pillows	Sofas	Tables	TV	Wardrobes
Average Quaternion Distance	0.1748	0.1091	0.1054	0.2653	0.2295	0.3828	0.1624	0.2553	0.1435	0.1350
Overall AQD	0.19631									

Table 4: Average Quaternion Distance Error results of all categories in the pose estimation evaluation test

Table 4 shows the average quaternion distance error among all 10 categories. The first interesting result we can see is that bottles and bookcases are the 2 categories that have the least distance error. We believe this is because these 2 classes do not change

orientations that much. Bottles is the more obvious example, because bottles are highly symmetrical, adjusting azimuth angles has visual no effect if the bottle is pointing straight up. Bookcases also have a limited possible rotation angle, because they are usually snapped to the wall with one prominent face pointing outwards. The opposite of this theory holds too, meaning objects that has a high freedom of rotation, often have higher quaternion distance error. For instance, chairs can freely rotate on the floor without much horizontal constraints. Therefore, it is common to see them in many orientations, and hence chairs have a relatively larger error. Averaging all the errors, we compute the mean quaternion distance across all 100 objects as 0.19631, which roughly translates to 23.75°. Looking at the results from [19], we notice that it can only produce object detection and pose estimation on large furniture (Bed, Wardrobe). In terms of error values, it computed an error of 27.3° for Beds and -16.7° for Wardrobes. Hence, we can see that our error value is actually not quite large comparatively. More importantly, as mentioned in the last section, we did not fine-tune the PoseFromShape model with our dataset. We believe the average quaternion distance error can be further lowered if the model is trained.

4.6.2 Pose Estimation Dataset

	Beds	Bookcases	Bottles	Chairs	Desks	Pillows	Sofas	Tables	TV	Wardrobes
Average IoU	0.7410	0.6672	0.8605	0.7000	0.6160	0.8964	0.7731	0.5691	0.8782	0.7433
Overall Average IoU	0.74442									

Table 5: Average IoU score of all categories in the dataset alignment quality test

(Table 5) shows the average IoU score between object mask and 2D projection of aligned CAD models. Overall, every category's score is quite satisfactory, especially for TvMonitors and Pillows, which have over 0.85. We suppose this is because these objects are quite rigid in shape and there are not many special contours. As opposed to shapes such as chairs and tables, their shapes and designs are highly varied, therefore

it is harder for 2D projections of CAD models to fully match object masks. Considering all the 100 objects, the overall average IoU score is 0.7444. As this dataset validation method is inspired by Pix3D, we compare our mean IoU score to it. We notice that their IoU can surpass 0.8 on chairs and 0.9 on sofas. We believe the reason behind Pix3D's high score is related to the high correspondence between its images and its 3D models. As they mentioned in their paper, they crawled images in search engines using IKEA furniture names as keywords, which means every image must correspond to a 3D model in their CAD pool. The high resemblance between 3D models and the images can greatly boost their overlap, which explains the high IoU scores. Comparing our results to other existing datasets, however, we find that our IoU for Chairs is significantly higher than that of ObjectNet3D, which has 0.570.

Chapter 5: Data Post-Processing and 3D Reconstruction

5.1 Introduction

The previous chapters have broken down the 3D scene reconstruction problem into individual subtasks, namely Room Layout Estimation, Object Detection, and Object Pose Estimation. In this section, we cover previous works specifically for 3D environment reconstruction from 2D images and explain how our approach aims to post-process data inferred from the previous steps into 3D scenes.

Although there were many previous works that focused on the aforementioned subtasks, there are not many complete approaches that fully address 3D scene reconstruction from images. Even if there are, these approaches suffer from a multitude of problems that render them inapplicable for VR applications development.

First of all, similar to layout estimation, the input to the reconstruction system is an important factor to the overall generated environment. Unfortunately, most of the past methods adopted perspective images as input, which have a FOV too small to capture all contextual information about the indoor environment [15, 16, 48, 58, 86 - 91]. Typically, the virtual environments generated from perspective images are bound to be incomplete. However, some approach tried to interpolate the room layout that is not captured by the input image [91], but it is impossible to verify if the estimated layout is valid, and the extended layout section would lack texture and color. In any case, if these incomplete environments are deployed in VR applications, users' immersive experience would most probably be negatively affected.

Second of all, the use of image depth data is another crucial factor. Some existing reconstruction approaches use RGB-D images as input to the system [15, 58, 88, 89]. Although depth information can help estimate object scale and distance of layout surfaces to camera, RGB-D images are less accessible than normal RGB images, and

hence these methods are not the most versatile options to aid virtual environment creation.

Last but not least, environment visual representation contributes greatly to the quality of virtual environment generation. For 3D scenes to immerse users and appear more realistic, room layouts and objects should be visually similar to their real life counterpart. However, many approaches use preliminary representations such as point cloud or even 3D bounding boxes to display components in the virtual environment. This can completely shatter the realness of the environment. Worse still, the problem propagates to panorama image inputs, where some approaches use similar representation techniques to display virtual environments.

Although our approach addresses all the above issues, we have to post-process CNN data output and convert them into 3D scenes. Specifically, we need to estimate the 3D position of the detected objects. Due to the lack of closely related approaches, we decided to utilize layout context and 2D object mask to programmatically compute it. Experiments show that this method provides satisfactory qualitative results, and act as a good starting point for further improvement.

5.2 Related Works

Similar to room layout, 3D scene reconstruction approaches can also be coarsely summarized as Perspective image approach [15, 16, 48, 58, 86 - 91], and Panorama image approach [17, 18, 19, 50, 53]. Among methods with perspective image input, we can further separate existing approaches as RGB-based, or RGB-D-based. An example of RGB-based approach is IM2CAD [10], where it utilized CNN for most of the system modules, from room layout estimation to 3D position estimation. Some approaches adopted image depth information to assist in reconstruction. For instance, perspective RGB and depth map were taken as input to produce point cloud environment, and multiple point cloud can be stitched together to produce a complete 3D scene. While it cannot offer any visualization of 3D objects, it did produce a fully semantic segmented 3D environment [88]. It is worth noting that this approach utilized a stream of perspective images, a video, to describe the entire indoor room context in order to

compensate for the lost contextual information of 1 single perspective image, which is an alternative of using panorama images as input. As discussed in the last subsection, perspective-based approach suffers from incomplete or uncertain environment context beyond the FOV of the camera, which lower the quality of generated 3D scenes. Apart from loss of context, some perspective approaches visualize 3D scene in representation which is too preliminary for VR application. For example, objects were expressed using wired 3D bounding boxes [90, 91], or even voxels [48]. In terms of appearance, these representations are either too preliminary, or too sophisticated compared to real life objects.

Despite the dominance in perspective-approach, there are a few previous works regarding panorama input. One of the most significant is PanoContext [17], where it used GC and OM to generate room layout and used rectangle detector and segmentation cuboid hypothesis to predict 3D objects. Although every surface in the generated environment is painted with texture, and is satisfactory for VR application, its largest limitation, in terms of visual representation, is that every 3D object is constrained to be cuboid shape, which implies strictly 6 surfaces. Furthermore, objects' orientation is presumed to be axis-aligned, which means objects can only be parallel/perpendicular to layout surfaces, and 3D pose cannot be slanted. This limitation is also apparent in [53], which estimates object cuboid lines using RANSAC and align their rotation to principal axes of the room. This orientation assumption is sensible to an extent, because most furniture are usually snapped to edges of the room to save space, but this does not usually apply to smaller scale objects such as chairs and TV Monitors. This is also the reason we decided to adopt PoseFromShape's deep learning approach to learn object pose.

[19] is the most related approach compared to ours. It divided panorama image to put in CNN for object detection, and applied template-matching to simultaneously retrieve CAD model and estimate 3DOF pose. For 3D position, it uses the pixel location estimated with Faster R-CNN and converts it in 3D coordinates. Our approach differs the most at 3D position estimation, because we decided to combine layout surface context generated by HorizonNet to aid object placement in the 3D scene.

5.3 Methodology

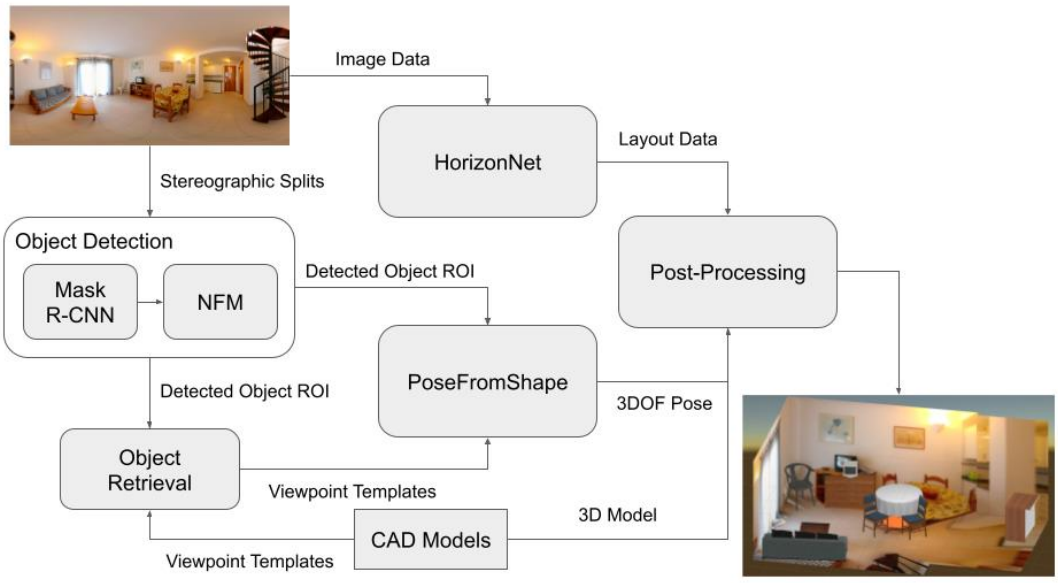

Fig. 18: The main structure of our 3D reconstruction system

This subsection mainly reiterates our entire 3D reconstruction system and elaborates on our data post-processing stage after the 3 main reconstruction stages.

First, we review the whole workflow of our 3D reconstruction system. Panorama image input is simultaneously put into HorizonNet and Mask R-CNN for room layout estimation and object detection respectively. Specifically, the panorama is divided into 4 sections before being individually inputted into Mask R-CNN. 3D corner coordinates and surface texture RGB data from HorizonNet will be passed to Unity for mesh room layout generation. At the same time, we obtain inferred object masks, ROI bounding boxes, and class labels of all splits from Mask R-CNN, and we execute Negative Feedback Mechanism to minimise misdetections and duplicate detections. After all objects from the panorama are detected, cropped ROI object masks will be sent to our template-based object retrieval system to match detected object with the best CAD model. When all object-shape pairs are established, they will be inputted into PoseFromShape and objects' 3DOF pose will be estimated in terms of azimuth,

elevation and in-plane rotation angles. Fig. 18 illustrates our whole reconstruction system approach.

Up to this stage, the only data missing is the 3D position of every detected object. We compute this through the output of our NFM and using layout context estimated from HorizonNet. We first explain our idea to the 3D position recovery algorithm, then elaborate further through implementation details:

For every detected object, we obtain the equirectangular reprojection of their object mask, and record its lowest pixel coordinates. This coordinate is crucial because depending on which surface it lies on, we can roughly estimate the visibility of the object, and even directly predict the object's 3D location. If the lowest point of the object lies on the floor plane, assuming the object's bottom part is not occluded, we can confirm that the bottom part of the 3D model is on the floor, and therefore we can directly align its retrieved 3D model's bottom part to the 3D position of the lowest pixel coordinates. If the lowest pixel coordinates are instead on the wall surfaces, we further query its estimated class. Typically, we assume only bottles, tv monitors, and pillows can possibly have location above ground. If so, we adjust their y axis position to be on top of its bottom object. If not, we can safely assume that the object in question is occluded by other objects, and we ask the user to manually align the object.

We now elaborate the implementation details of the 3D position recovery mechanism. Recall that we obtained the "center angles" for every detected object through the NFM. We define an object's center angles as the yaw and pitch angles required to be rotated if the center point of the object is to be placed at the center of the original panorama. We then convert center angles to the pixel coordinates of object center relative to input equirectangular panorama. This center coordinates are used as reference to coarsely predict where the object will locate in the 3D scene. The next step is to reproject the object center mask to the original equirectangular projection. Recall that we also obtained the center mask of detected objects, which is the object mask when the detected object is centered under stereographic projection. We convert the mask back to equirectangular projection. Here we address this reprojected mask as "EquiObjMask". After that, we compute the visually lowest pixel coordinates of the

EquiObjMask. For objects such as tables and chairs, this would be the pixel coordinates of their legs. After the lowest pixel is obtained, we cross-reference it to the surface segmentation mapping produced by HorizonNet, in order to determine which surface it belongs to. If the pixel lies on the floor plane, we snap the 3D model directly to its 3D position. The position alignment is executed in Unity, which excels in manipulating object transformation. For the rotation angles of the objects, we put a virtual camera in the origin position of the Unity 3D scene and point it to the object 3D location. According to the estimated 3DOF pose from PoseFromShape, we rotate the object by the same angles. If the object appears to be "floating" above ground because of inference error, we tweak the elevation angle so that the bottom part of the object is firmly stuck to the ground.

5.4 Experiments

In order to analyze the performance of the reconstruction output, we conduct the following experiment.

For the dataset, we use the modified SUN360Ext dataset similar to the experiment we conducted on Mask R-CNN. We randomly select 50 equirectangular test images from the test set and input them into the whole reconstruction system. The images will go through object detection, pose estimation and post processing before displaying in a mesh-based Unity scene. We then manually go through the generated environments and pick the most significant one for qualitative evaluation. We define "the most significant scene" as having the most common properties that most reconstructed 3D scenes have. It should also be a generated environment which can effectively represent all the others.

5.5 Evaluations and Results

5.5.1 Evaluation Methods

5.5.1.1 Quantitative analysis

Note that we do not evaluate generated 3D scenes quantitatively, for the following reasons.

First of all, there are only a very limited number of datasets that can evaluate complete 3D indoor scenes. SUN RGB-D [90] is one of the available datasets, but it contains perspective images input only. Another example is Panocontext [17], but as previously mentioned, its object annotations are cuboid-shaped instead of authentic CAD models with unique contours. We believe that using this dataset for evaluation will produce inaccurate results.

Second of all, we already conducted extensive quantitative evaluation of the previous chapters, which are subcomponents of the entire system. Considering that the whole post-processing phase is purely programmatic, we believe we can assume the whole reconstruction's quantitative performance based on its submodules.

Last but not least, we would like to emphasize that the current system mainly aims to aid modelers by speeding up the environment generation process. Despite the possibility that the generated environments are not entirely accurate, it still accelerated the design flow. Therefore, we propose that precise quantitative analysis in this chapter is not necessary.

5.5.1.2 Qualitative Analysis

We select a most significant generated scene as an demonstration, and assess the environment based on successful detection or pose estimations and possible failed cases.

5.5.2 Results

Fig. 19 shows the input image, as well as the reconstructed environment. We first analyze the successful instances of the environment. First of all, we made use of HorizonNet to generate the indoor 3D layout, which works really well under most of the layouts. Non-cuboid room layouts can be effectively estimated. In terms of object detection, we can see their cases where objects are successfully detected using Mask R-CNN and NFM, such as the round table, bookcase and tvmonitor. However, objects which are under heavy occlusion, for example the 2 chairs behind the round table, are not detected. We consider these cases acceptable, because even though these objects are detected, we do not have sufficient visual data about them to properly retrieve CAD or estimate its 3DOF pose. So, we leave them to the user to manually add them in the scene.

In terms of object pose estimation, we can see angular errors being visualized in many objects. For example, the sofa's predicted rotation is not entirely aligned with the wall. The most extreme failure cases here are the two chairs near the round table. In the input image, we can see they need to face inwards towards the table, but the system predicted a quite visible error. We believe this is due to them being occluded and facing an unusual direction (backwards to the camera).

In terms of post-processing and 3D position estimation, we think the 3D positions are satisfactory, with the objects being near its "silhouette" on the room layout. We can also see I restricted pillows position to follow sofas/beds, as seen in the pillows at lower left. Also, we added collision detection between objects, as well as between surface and objects, so we can see no objects or geometry passing through each other.

Overall, although object rotations are not perfect, we think our reconstruction system provides good coarse estimations for what is in the indoor environment, and their respective 6DOF poses.

Fig. 19: Example of input panorama and its reconstructed 3D scene displayed in Unity

Chapter 6: Conclusion and Future Work

6.1 Conclusion

In this thesis, we proposed a CNN based 3D reconstruction system that takes 360 equirectangular panoramas, and converts them into their corresponding 3D scenes, and displays them in the Unity Game Engine. The system can be broken down into 4 core subtasks, namely Room Layout Estimation, Object Detection, Object Pose Estimation, and Artistic Postprocessing. Compared to existing reconstruction systems, our approach is capable of generating non-cuboid room layouts and retrieve suitable realistic CAD models to represent objects in 3D scenes. We adopted Deep Neural Networks, especially CNNs to learn visual features from panorama images and make predictions regarding different aspects in the environment.

As our goal of creating this system is to aid 3D modelers by speeding up VR virtual environments design progress, we prioritized the compatibility between VR applications and our approach. For room layout estimation, we converted HorizonNet's point cloud layout to mesh representation. Experiment showed the conversion makes room layouts visually more appealing, improves frame rate and optimized memory usage.

In order to train the pose estimation network such that it specializes in indoor pose predictions, we constructed a custom dataset with a few inspirations from existing datasets. Compared to these past datasets, we managed to diversify image source, and made enhancement by combining unique annotation features from all previous methods into our own dataset. Last but not least, we created a Unity based annotation tool for annotators to label their custom data or even enhance our version of the dataset.

We believe our 3D reconstruction system can be mainly utilized in the VR development industry, in order to lower the time complexity of creating virtual environment from scratch. 3D modelers can take a panorama image with his/her mobile device and directly obtain an estimation of the 3D environment. If the modeler is not satisfied with the details, such as 3D poses, they can tweak the model fairly easily because the 3D

scene can be freely modified in Unity once it is created. Also, Since Unity supports VR development tools both natively or by third-party plugins, developers can start VR application development with the generated environment without needing to switch software or development platform.

Apart from professional developers, this system can also allow laymen content creators to quickly mock up a 3D environment, even without any knowledge in 3D modeling. In other words, our system can make 3D content creation more readily available to the general public.

6.2 Future Work

Although our current approach is able to smoothly convert panorama images into 3D virtual environments, the accuracy is far from perfect. Here we propose two future works that can be further investigated.

Our pose estimation network is perhaps one of the regions that needs to be improved. One feasible approach is to obviously fine-tune PoseFromShape with our own dataset in order to make sure it fits indoor room objects better than the original ObjectNet3D weights. We believe the quaternion distance error can be reduced using this method.

The other area for potential improvement is object retrieval. In our current approach, we use a simple IoU matching algorithm to compare 3D shapes between detected objects ROI and different rendered viewpoints of 3D CAD models. Using IoU as the metric is not ideal because it will neglect interior features, for example the compartments in bookcases, during comparison. An approach of improvement is to incorporate the ROI feature layer of feature extraction CNNs, so that visual features of detected objects, as well as the rendered images, can be compared thoroughly before applying metric to measure similarity.

Bibliography

[1] Coughlan, J. M., & Yuille, A. L. (2000). The Manhattan world assumption: Regularities in scene statistics which enable Bayesian inference. In T. Leen, T. Dietterich, & V. Tresp (Eds.), Advances in neural information processing systems (Vol. 13, pp. 845-851). Neural Information Processing Systems Foundation. https://papers.nips.cc/paper/2000/file/90e1357833654983612fb05e3ec9148c-Paper.pdf

[2] Delage, E., Lee, H., & Ng, A. Y. (2006). A dynamic Bayesian network model for autonomous 3D reconstruction from a single indoor image. In A. Fitzgibbon, C. J. Taylor, & Y. LeCun (Eds.), 2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06) (Vol. 2, pp. 2418-2428). IEEE. doi: 10.1109/cvpr.2006.23

[3] Lee, D. C., Hebert, M., & Kanade, T. (2009). Geometric reasoning for single image structure recovery. 2009 IEEE conference on computer vision and pattern recognition (pp. 2136-2143). IEEE. doi: 10.1109/cvpr.2009.5206872

[4] Hedau, V., Hoiem, D., & Forsyth, D. (2009). Recovering the spatial layout of cluttered rooms. 2009 IEEE 12th international conference on computer vision (pp. 1849-1856). IEEE. doi: 10.1109/iccv.2009.5459411

[5] Mallya, A., & Lazebnik, S. (2015). Learning informative edge maps for indoor scene layout prediction. 2015 IEEE international conference on computer vision (ICCV) (pp. 936-944). IEEE. doi: 10.1109/iccv.2015.113

[6] Zhang, W., Zhang, W., & Gu, J. (2019). Edge-semantic learning strategy for layout estimation in indoor environment. IEEE Transactions on Cybernetics, 50(6), 2730-2739. doi: 10.1109/tcyb.2019.2895837

[7] Zou, C., Colburn, A., Shan, Q., & Hoiem, D. (2018). LayoutNet: Reconstructing the 3D room layout from a single RGB image. 2018 IEEE/CVF conference on

computer vision and pattern recognition (pp. 2051-2059). IEEE. doi: 10.1109/cvpr.2018.00219

[8] Lee, C. Y., Badrinarayanan, V., Malisiewicz, T., & Rabinovich, A. (2017). RoomNet: End-to-end room layout estimation. 2017 IEEE international conference on computer vision (ICCV) (pp. 4865-4874). IEEE. doi: 10.1109/iccv.2017.521

[9] Yang, S. T., Wang, F. E., Peng, C. H., Wonka, P., Sun, M., & Chu, H. K. (2019). DuLa-Net: A dual-projection network for estimating room layouts from a single RGB panorama. 2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR) (pp. 3363-3372). IEEE. doi: 10.1109/cvpr.2019.00348

[10] Izadinia, H., Shan, Q., & Seitz, S. M. (2017). IM2CAD. 2017 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 5134-5143). IEEE. doi: 10.1109/cvpr.2017.260

[11] Fernandez-Labrador, C., Facil, J. M., Perez-Yus, A., Demonceaux, C., Civera, J., & Guerrero, J. J. (2020). Corners for layout: End-to-end layout recovery from 360 images. IEEE Robotics and Automation Letters, 5(2), 1255-1262. doi: 10.1109/lra.2020.2967274

[12] Sun, C., Hsiao, C. W., Sun, M., & Chen, H. T. (2019). HorizonNet: Learning room layout with 1d representation and pano stretch data augmentation. 2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR) (pp. 1047-1056). IEEE. 10.1109/cvpr.2019.00114

[13] Zhang, J., Kan, C., Schwing, A. G., & Urtasun, R. (2013). Estimating the 3D layout of indoor scenes and its clutter from depth sensors. 2013 IEEE international conference on computer vision (pp. 1273-1280). IEEE. doi: 10.1109/iccv.2013.161

[14] Silberman, N., Hoiem, D., Kohli, P., & Fergus, R. (2012). Indoor segmentation and support inference from RGDB images. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, & C. Schmid (Eds.), Lecture notes in computer science: Vol. 7576. ECCV 2012: Computer vision – ECCV 2012 (pp. 746-760). Springer. doi: 10.1007/978-3-642-33715-4_54

[15] Guo, R., Zou, C., & Hoiem, D. (2015). Predicting complete 3D models of indoor scenes. arXiv preprint. https://arxiv.org/abs/1504.02437

[16] Tulsiani, S., Gupta, S., Fouhey, D. F., Efros, A. A., & Malik, J. (2018). Factoring shape, pose, and layout from the 2D image of a 3D scene. 2018 IEEE/CVF conference on computer vision and pattern recognition (pp. 302-310). IEEE. doi: 10.1109/cvpr.2018.00039

[17] Zhang, Y., Song, S., Tan, P., & Xiao, J. (2014). PanoContext: A whole-room 3D context model for panoramic scene understanding. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), Lecture notes in computer science: Vol. 8694. ECCV 2014: Computer vision – ECCV 2014 (pp. 668-686). Springer. doi: 10.1007/978-3-319-10599-4_43

[18] Xu, J., Stenger, B., Kerola, T., & Tung, T. (2017). Pano2CAD: Room layout from a single panorama image. In 2017 IEEE winter conference on applications of computer vision (WACV) (pp. 354-362). IEEE. doi: 10.1109/wacv.2017.46

[19] Luo, C., Zou, B., Lyu, X., & Xie, H. (2019). Indoor scene reconstruction: From panorama images to CAD models. 2019 IEEE international symposium on mixed and augmented reality adjunct (ISMAR-Adjunct) (pp. 317-320). IEEE. doi: 10.1109/ismar-adjunct.2019.00-21

[20] Schubert, T., Friedmann, F., & Regenbrecht, H. (1999). Embodied presence in virtual environments. In R. Paton & I. Neilson (Eds.) Visual representations and interpretations (pp. 269-278). Springer. doi: 10.1007/978-1-4471-0563-3_30

[21] Bohil, C., Owen, C., Jeong, E., Alicea, B., & Biocca, F. (2009). Virtual reality and presence. In W. F. Eadie (Ed.), 21st century communication: A reference handbook (Vol 2, pp. 534-542). SAGE Publications. doi: 10.4135/9781412964005.n59

[22] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. 2016 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 770-778). IEEE. doi: 10.1109/cvpr.2016.90

[23] Graves, A. (2012). Long short-term memory. Studies in computational intelligence: Vol. 385. Supervised sequence labelling with recurrent neural networks (pp. 37-45). Springer. doi: 10.1007/978-3-642-24797-2_4

[24] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11), 2673-2681. doi: 10.1109/78.650093

[25] Xiao, J., Ehinger, K. A., Oliva, A., & Torralba, A. (2012). Recognizing scene viewpoint using panoramic place representation. 2012 IEEE conference on computer vision and pattern recognition (pp. 2695-2702). IEEE. doi: 10.1109/cvpr.2012.6247991

[26] Armeni, I., Sax, S., Zamir, A. R., & Savarese, S. (2017). Joint 2D-3D-semantic data for indoor scene understanding. arXiv preprint. https://arxiv.org/abs/1702.01105

[27] Unity. (2019). Unity game engine (Version 2018.4.6) [Computer software]. https://unity3d.com/unity/whats-new/2018.4.6

[28] Unity. (2019). Triangulator [Computer script]. Unity. https://wiki.unity3d.com/index.php/triangulator

[29] Votanic. (2021). VotanicXR (Version 1.0.9) [Computer script]. https://www.votanic.com/votanicxr

[30] Virtanen, J. P., Daniel, S., Turppa, T., Zhu, L., Julin, A., Hyyppä, H., & Hyyppä, J. (2020). Interactive dense point clouds in a game engine. ISPRS Journal of Photogrammetry and Remote Sensing, 163, 375-389. doi: 10.1016/j.isprsjprs.2020.03.007

[31] Chan, J. C. P., Ng, A. K. T., & Lau, H. Y. K. (2021). Constructing 3D mesh indoor room layouts from 2D equirectangular RGB 360 panorama images for the Unity game engine. In C. Stephanidis, M. Antona, & S. Ntoa (Eds.), Communications in computer and information science: Vol. 1421. HCI international 2021 - Posters (pp. 148–155). Springer. doi: 10.1007/978-3-030-78645-8_19

[32] Meehan, M., Insko, B., Whitton, M., & Brooks Jr, F. P. (2002). Physiological measures of presence in stressful virtual environments. ACM Transactions on Graphics, 21(3), 645-652. doi: 10.1145/566654.566630

[33] Kolasinski, E. M. (1995). Simulator sickness in virtual environments (Vol. 1027). US Army Research Institute for the Behavioral and Social Sciences. https://apps.dtic.mil/sti/citations/ADA295861

[34] Pane, Martin. [Tayx] (2021). [Graphy] - Ultimate FPS Counter - Stats Monitor & Debugger (Version 2.1.3) [Computer software]. https://assetstore.unity.com/packages/tools/gui/ graphy-ultimate-fps-counter-stats-monitor-debugger-105778

[35] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2), 91-110. doi: 10.1023/b:visi.0000029664.99615.94

[36] Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In C. Schmid, S. Soatto, & C. Tomasi (Eds.), 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05) (Vol. 1, pp. 886-893). IEEE. doi: 10.1109/cvpr.2005.177

[37] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297. doi: 10.1007/bf00994018

[38] Cunningham, P., & Delany, S. J. (2020). k-nearest neighbour classifiers: (with Python examples). arXiv preprint. https://arxiv.org/abs/2004.04523

[39] Lee, A. (2015). Comparing deep neural networks and traditional vision algorithms in mobile robotics. Swarthmore University. https://www.cs.swarthmore.edu/~meeden/cs81/f15/papers/Andy.pdf

[40] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), Advances in neural information processing systems (Vol. 25,

pp. 1097-1105). Neural Information Processing Systems Foundation. https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[41] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. 2014 IEEE conference on computer vision and pattern recognition (pp. 580-587). IEEE. doi: 10.1109/cvpr.2014.81

[42] Girshick, R. (2015). Fast R-CNN. 2015 IEEE international conference on computer vision (ICCV) (pp. 1440-1448). IEEE. doi: 10.1109/iccv.2015.169

[43] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), Advances in neural information processing systems (Vol. 28, pp. 91-99). Neural Information Processing Systems Foundation. https://papers.nips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

[44 He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(2), 386-397. doi: 10.1109/tpami.2018.2844175

[45] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. 2016 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 779-788). IEEE. doi: 10.1109/cvpr.2016.91

[46] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds), Lecture notes in computer science: Vol. 9905. ECCV 2016: Computer vision – ECCV 2016. (pp. 21-37) Springer. doi: 10.1007/978-3-319-46448-0_2

[47] Dvornik, N., Shmelkov, K., Mairal, J., & Schmid, C. (2017). BlitzNet: A real-time deep network for scene understanding. 2017 IEEE international conference on computer vision (ICCV) (pp. 4154-4162). IEEE. doi: 10.1109/iccv.2017.447

[48] Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., & Funkhouser, T. (2017). Semantic scene completion from a single depth image. 2017 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 1746-1754). IEEE. doi: 10.1109/cvpr.2017.28

[49] Gupta S., Girshick R., Arbeláez P., & Malik J. (2014) Learning rich features from RGB-D images for object detection and segmentation. In Fleet D., Pajdla T., Schiele B., & Tuytelaars T. (Eds), Lecture notes in computer science: Vol. 8695. ECCV 2014: Computer vision – ECCV 2014 (pp. 345-360). Springer. doi: 10.1007/978-3-319-10584-0_23

[50] Yang, Y., Jin, S., Liu, R., Kang, S. B., & Yu, J. (2018). Automatic 3D indoor scene modeling from single panorama. 2018 IEEE/CVF conference on computer vision and pattern recognition (pp. 3926-3934). IEEE. doi: 10.1109/cvpr.2018.00413

[51] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint. https://arxiv.org/abs/1409.1556

[52] Yang, W., Qian, Y., Kämäräinen, J. K., Cricri, F., & Fan, L. (2018). Object detection in equirectangular panorama. 2018 24th international conference on pattern recognition (ICPR) (pp. 2190-2195). IEEE. doi: 10.1109/icpr.2018.8546070

[53] Guerrero-Viu, J., Fernandez-Labrador, C., Demonceaux, C., & Guerrero, J. J. (2020). What's in my room? Object recognition on indoor panoramic images. 2020 IEEE international conference on robotics and automation (ICRA) (pp. 567-573). IEEE. doi: 10.1109/icra40945.2020.9197335

[54] Chou, S. H., Sun, C., Chang, W. Y., Hsu, W. T., Sun, M., & Fu, J. (2020). 360-Indoor: Towards learning real-world objects in 360° indoor equirectangular images. 2020 IEEE winter conference on applications of computer vision (WACV) (pp. 845-853). IEEE. doi: 10.1109/wacv45572.2020.9093262

[55] Wang, K. H., & Lai, S. H. (2019). Object detection in curved space for 360-degree camera. ICASSP 2019-2019 IEEE international conference on acoustics, speech and

signal processing (ICASSP) (pp. 3642-3646). IEEE. doi: 10.1109/icassp.2019.8683093

[56] Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. In A. Leonardis, H. Bischof, & A. Pinz (Eds), Lecture notes in computer science: Vol. 3951. Computer vision – ECCV 2006. ECCV 2006 (pp. 404-417). Springer. doi: 10.1007/11744023_32

[57] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds), Lecture notes in computer science: Vol. 8693. Computer vision – ECCV 2014. ECCV 2014 (pp. 740-755). Springer. doi: 10.1007/978-3-319-10602-1_48

[58] Zou, C., Guo, R., Li, Z., & Hoiem, D. (2019). Complete 3D scene parsing from an RGBD image. International Journal of Computer Vision, 127(2), 143-162. doi: 10.1007/s11263-018-1133-z

[59] Xiang, Y., Kim, W., Chen, W., Ji, J., Choy, C., Su, H., ... & Savarese, S. (2016). ObjectNet3D: A large scale database for 3D object recognition. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds), Lecture notes in computer science: Vol. 9912. Computer vision – ECCV 2016. ECCV 2016. (pp. 160-176). Springer. doi: 10.1007/978-3-319-46484-8_10

[60] Mottaghi, R., Xiang, Y., & Savarese, S. (2015). A coarse-to-fine model for 3D pose estimation and sub-category recognition. 2015 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 418-426). IEEE. doi: 10.1109/cvpr.2015.7298639

[61] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). IEEE. doi: 10.1109/cvpr.2009.5206848

[62] Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2018). Focal Loss for Dense Object Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(2), 318-327. doi: 10.1109/tpami.2018.2858826

[63] The Blender Foundation. (2016). Blender (Version 2.77) [Computer software]. https://www.blender.org/download/releases/2-77

[64] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., & Navab, N. (2012). Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In K. M. Lee, Y. Matsushita, J. M. Rehg, & Z. Hu (Eds), Lecture notes in computer science: Vol. 7724. ACCV 2012: Computer vision – ACCV 2016. (pp. 548-562) Springer. doi: 10.1007/978-3-642-37331-2_42

[65] Wohlhart, P., & Lepetit, V. (2015). Learning descriptors for object recognition and 3D pose estimation. 2015 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 3109-3118). IEEE. doi: 10.1109/cvpr.2015.7298930

[66] Bansal, A., Russell, B., & Gupta, A. (2016). Marr revisited: 2D-3D alignment via surface normal prediction. 2016 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 5965-5974). IEEE. doi: 10.1109/cvpr.2016.642

[67] Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6), 381-395. doi: 10.1145/358669.358692

[68] Grabner, A., Roth, P. M., & Lepetit, V. (2018). 3D pose estimation and 3D model retrieval for objects in the wild. 2018 IEEE/CVF conference on computer vision and pattern recognition (pp. 3022-3031). IEEE. doi: 10.1109/cvpr.2018.00319

[69] Tekin, B., Sinha, S. N., & Fua, P. (2018). Real-time seamless single shot 6D object pose prediction. 2018 IEEE/CVF conference on computer vision and pattern recognition (pp. 292-301). IEEE. doi: 10.1109/cvpr.2018.00038

[70] Georgakis, G., Karanam, S., Wu, Z., & Kosecka, J. (2019) Learning local RGB-to-CAD correspondences for object pose estimation. 2019 IEEE/CVF international

conference on computer vision (ICCV) (pp. 8966-8975). IEEE. doi: 10.1109/iccv.2019.00906

[71] Pavlakos, G., Zhou, X., Chan, A., Derpanis, K. G., & Daniilidis, K. (2017). 6-DoF object pose from semantic keypoints. 2017 IEEE international conference on robotics and automation (ICRA) (pp. 2011-2018). IEEE. doi: 10.1109/icra.2017.7989233

[72] Zakharov, S., Shugurov, I., & Ilic, S. (2019) DPOD: 6D pose object detector and refiner. 2019 IEEE/CVF international conference on computer vision (ICCV) (pp. 1941-1950). IEEE. doi: 10.1109/ICCV.2019.00203

[73] Tulsiani, S., & Malik, J. (2015). Viewpoints and keypoints. 2015 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 1510-1519). IEEE. doi: 10.1109/CVPR.2015.7298758

[74] Su, H., Qi, C. R., Li, Y., & Guibas, L. J. (2015). Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3S model views. 2015 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 2686-2694). IEEE. doi: 10.1109/iccv.2015.308

[75] Xiang, Y., Schmidt, T., Narayanan, V., & Fox, D. (2017). PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. arXiv preprint. https://arxiv.org/abs/1711.00199

[76] Wu, J., Zhou, B., Russell, R., Kee, V., Wagner, S., Hebert, M., ... & Johnson, D. M. (2018). Real-time object pose estimation with pose interpreter networks. 2018 IEEE/RSJ international conference on intelligent robots and systems (IROS) (pp. 6798-6805). IEEE. doi: 10.1109/iros.2018.8593662

[77] Xiao, Y., Qiu, X., Langlois, P. A., Aubry, M., & Marlet, R. (2019). Pose from shape: Deep pose estimation for arbitrary 3D objects. arXiv preprint. https://arxiv.org/abs/1906.05105

[78] Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., & Rother, C. (2014, September). Learning 6D object pose estimation using 3D object coordinates.

In Fleet D., Pajdla T., Schiele B., & Tuytelaars T. (Eds), Lecture notes in computer science: Vol. 8690. ECCV 2014: Computer vision – ECCV 2014 (pp. 536-551). Springer. doi: 10.1007/978-3-319-10605-2_35

[79] Xiang, Y., Mottaghi, R., & Savarese, S. (2014). Beyond pascal: A benchmark for 3D object detection in the wild. IEEE winter conference on applications of computer vision (pp. 75-82). IEEE. doi: 10.1109/wacv.2014.6836101

[80] Sun, X., Wu, J., Zhang, X., Zhang, Z., Zhang, C., Xue, T., ... & Freeman, W. T. (2018). Pix3D: Dataset and methods for single-image 3D shape modeling. 2018 IEEE/CVF conference on computer vision and pattern recognition (pp. 2974-2983). IEEE. doi: 10.1109/cvpr.2018.00314

[81] Lim, J. J., Pirsiavash, H., & Torralba, A. (2013). Parsing IKEA objects: Fine pose estimation. 2013 IEEE international conference on computer vision (pp. 2992-2999). IEEE. doi: 10.1109/iccv.2013.372

[82] Yu, F., Koltun, V., & Funkhouser, T. (2017). Dilated Residual Networks. 2017 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 636-644). IEEE. doi: 10.1109/cvpr.2017.75

[83] Trimble. (2019). 3D warehouse [3D model library]. http://3dwarehouse.sketchup.com

[84] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., ... & Yu, F. (2015). ShapeNet: An information-rich 3D model repository. arXiv preprint https://arxiv.org/abs/1512.03012

[85] Huang S., Qi S., Zhu Y., Xiao Y., Xu Y., Zhu SC. (2018) Holistic 3D scene parsing and reconstruction from a single RGB image. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds), Lecture notes in computer science: Vol. 11211. ECCV 2018: Computer vision – ECCV 2018 (pp. 194-211). Springer. doi: 10.1007/978-3-030-01234-2_12

[86] Guo, R., & Hoiem, D. (2013). Support surface prediction in indoor scenes. 2013 IEEE international conference on computer vision (pp. 2144-2151). IEEE. doi: 10.1109/iccv.2013.266

[87] Zhao, C., Sun, L., & Stolkin, R. (2017). A fully end-to-end deep learning approach for real-time simultaneous 3D reconstruction and material recognition. 2017 18th international conference on advanced robotics (ICAR) (pp. 75-82). IEEE. doi: 10.1109/icar.2017.8023499

[88] Schwing, A. G., Fidler, S., Pollefeys, M., & Urtasun, R. (2013). Box in the box: Joint 3D layout and object reasoning from single images. 2013 IEEE international conference on computer vision (pp. 353-360). IEEE. doi: 10.1109/iccv.2013.51

[89] Choi, W., Chao, Y. W., Pantofaru, C., & Savarese, S. (2013). Understanding Indoor Scenes Using 3D Geometric Phrases. 2013 IEEE conference on computer vision and pattern recognition (pp. 33-40). IEEE. doi: 10.1109/cvpr.2013.12

[90] Song, S., Lichtenberg, S. P., & Xiao, J. (2015). SUN RGB-D: A RGB-D scene understanding benchmark suite. 2015 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 567-576). IEEE. doi: 10.1109/cvpr.2015.7298655