Learning to Model 3D Indoor Scenes and Articulated Objects

by

Akshay Gadi Patil

M.Tech., Indian Institute of Technology Gandhinagar, 2016
B.E., P.E.S Institute of Technology, 2012

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in the School of Computing Science Faculty of Applied Sciences

© Akshay Gadi Patil 2023
SIMON FRASER UNIVERSITY
Summer 2023

Copyright in this work is held by the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.
Declaration of Committee

Name: Akshay Gadi Patil

Degree: Doctor of Philosophy

Thesis title: Learning to Model 3D Indoor Scenes and Articulated Objects

Committee:

Hao (Richard) Zhang
Supervisor
Professor, Computing Science

Manolis Savva
Committee Member
Assistant Professor, Computing Science

Daniel Cohen-Or
Examiner
Adjunct Professor, Computing Science

Qixing Huang
External Examiner
Associate Professor, Computer Science
The University of Texas at Austin
Abstract

Indoor environments are mainly composed of man-made, functional 3D objects plausibly arranged in a region-bounded space. Arguably, the most important goal when designing an indoor scene is to make it functional, meaning that the resulting scene should serve its intended usage. For example, a living room is typically comprised of a sofa set and a TV, serving the usage of "watching TV", and their arrangement in the room should not constrain movement and space usage. This is determined, in parts, by object relations and their co-occurrences, as well as the different functionalities and interactions offered by the individual objects themselves. Object functionality involves part-level reasoning, where object parts can undergo motions/articulations. Furniture models typically comprising indoor scenes inevitably reveal their interiors when undergoing part articulations. For example, one can observe the interiors when a cabinet door is opened. In other words, the overall scene layout, in terms of object arrangements, as well as the objects themselves, in terms of their part motions and geometry, must serve the intended purposes.

Developing computational tools for designing such indoor scenes is challenging because the desired solution(s) to the end goal can not always be simplified, if at all, to a set of simple rules. With the availability of large datasets and appropriate computational resources, it is natural to seek data-driven learning-based algorithms to model these entities. This dissertation explores the design of 3D indoor scenes, with advanced algorithmic development and evaluation in mind, going from the scene layout level to the object level, where functionality plays a key role in both cases. To this end, the dissertation is made up of three works, the first of which, GRAINS, presents the first end-to-end deep generative hierarchical neural network that learns object relations and co-occurrences following indoor object arrangement rules, and synthesizes novel 3D scenes. There is however a lack of principled means to evaluate and compare the generated scenes, which leads to the next work, LayoutGMN. As a first step, it presents a neural graph matching network that compares abstractions of 3D indoor scenes in a structural manner. Finally, for understanding and modeling functionality at the object level, a neural framework, called RoSI, for recovering 3D shape interiors and realizing 3D part motions from sparse multi-articulation images of 3D shapes is presented.

Keywords: 3D indoor scenes, Interactable 3D shapes, Structural similarity, Neural Nets
Dedication

To my Parents, Grandparents, and to my Wife
Acknowledgements

For all the things that have happened to me and the energy that has been with me during this period, I thank the almighty for this and the strength bestowed upon me to face all the challenges, for which I shall remain indebted to him, forever.

Few things in life evoke the kind of feelings on a personal level as the ones when writing this section. I want to start by first thanking my advisor Dr. Hao (Richard) Zhang. There are few people who have sound technical basics and pay a humongous amount of attention to details. And then there are fewer people who, in addition to the previous two qualities, also have the ability to envision and realize new research directions. Richard is one of those, and these things seem to come naturally to him. What is more inspiring is the incredible way he works despite being the sandwich generation in his family. This section will overflow if I continue, so let me stop. It can only be termed a universe conspiracy if one gets to apprentice with a person whose attitude strongly complements your tenacity and inspires you to keep pushing your limits. Thank you for the opportunity, support, and trust you showed in me during this period.

Dr. Daniel Cohen-Or is another person to whom I owe my gratitude. His mentorship skills are unrivaled, and I can see why that is the case. His encouraging smile, composure, and stoic push to work relentlessly from the get-go are rare qualities, something that I have witnessed first-hand on two different projects. His friendly attitude is most endearing, which only made him more approachable. Thank you for constantly supporting me.

My Ph.D. journey took off with two concurrent publications, working closely with Manyi Li on both projects. I do not think one can find a project partner like Manyi. She is a wonderful collaborator, to say the least, who is always available to brainstorm, discuss and plow through various ideas to find solutions. Her amazing readiness to take on responsibility makes her an added asset. I am glad to have started off and continued my publications with you, and I am sure there are many more to come. Thank you for the fond memories, both when working on the projects and outside the lab. In the same spirit, I want to express my gratitude to Yiming Qian, with whom I collaborated closely during my internship at Amazon, Vancouver. He was always ready to help, however small the issue might be, both on and off research projects. Building a life in a new country from scratch shows his amazing
strength and the amount of hard work he puts in every day. His story is quite inspiring and I am glad to have met and worked with you. Thanks also to my internship mentors at Adobe, Vova, Sid, and Matt. Your problem-formulating and more importantly, critiquing skills, are something that I will always carry forward. I’ve also had the opportunity to spend time in Israel during my first internship at Amazon, and I would like to thank my friends and team members there, Nadav Schor, Or Perel, Roee, Omri, Hadar, and Shai, for their warmth and support. You guys were welcoming and enthusiastic in every aspect, which is something I will always remember fondly. Finally, thanks to some folks at GrUVi, Chenyang, Wallace, Kangxue and Luwei, who were comically innovative in their own ways, making things inside the lab less stressful and a lot more fun. The CS department at the school, especially the ever-changing face behind the role of Grad Program Asst., handled all our paperwork, making life easier. Thank you for the continued assistance.

Finally, I want to thank my beloved parents and grandparents for this life and for their unconditional support. I draw an unparalleled amount of inspiration from you all. I am truly fortunate and grateful for the amount of time you spent shaping my life in every developmental aspect, regardless of the difficulties and challenges you faced. All along, you were there to share my joy, sorrow and grow in excitement with me and comfort me when needed. My maternal grandparents have always served as an inspiration to me for their calmness in carrying forward the family and their unending love despite the lifelong struggles they faced. Your belief in me has only strengthened my conviction to pursue my dreams.

A very special person entered my life during this journey. A beautiful soul, you balance me in life and I am grateful for your unwavering support and blind faith in all my endeavors. The kind of memories I have accumulated with you in the last few years are beyond special. There is never a dull moment when you are around. For the many adventures we have been through, the many more to come, and for the journey together, Salud!

Home is where the heart is. I have been a fortunate beneficiary of the invaluable resources of my homeland, Bharat, making me capable of starting this journey in a new continent. My country, its armed forces, culture, and ethos will always be a vital part of me, and I hope to have the opportunity to contribute to its growth. Long live my homeland, India. I want to also express my heartfelt gratitude to Canada for the life here, which has been my home during this journey.
Table of Contents

Declaration of Committee ii
Abstract iii
Dedication iv
Acknowledgements v
Table of Contents vii
List of Tables x
List of Figures xii

1 Introduction 1
  1.1 Motivation ......................................................... 1
    1.1.1 Why model 3D indoor scenes? .............................. 2
    1.1.2 Why model articulated objects? .......................... 2
    1.1.3 Why learn to model them? ................................ 3
    1.1.4 Specific problems addressed .............................. 5
  1.2 Challenges ....................................................... 5
    1.2.1 Neural Generation of 3D Indoor Scenes .................. 5
    1.2.2 Learning Structural Similarity for 2D Floorplans ...... 6
    1.2.3 Learning to Recover 3D Shape Interiors from Few Articulation Images 7
  1.3 Overview of the thesis ......................................... 7
    1.3.1 Neural Generation of 3D Indoor Scenes .................. 7
    1.3.2 Neural Graph Matching for Structural Similarity of 2D Floorplans 8
    1.3.3 Recovering 3D Shape Interiors from Sparse Articulation Images 8
  1.4 Thesis contribution .............................................. 9
  1.5 Thesis organization ........................................... 10

2 Background 11
  2.1 Indoor 3D Scene Synthesis ................................. 11
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2 Learning Structural Similarity for 2D Floorplans</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Recovering 3D Shape Interiors and Part Motions from Articulation Images</td>
<td>16</td>
</tr>
<tr>
<td>3 Generative Recursive Autoencoders for Indoor Scenes</td>
<td>18</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>18</td>
</tr>
<tr>
<td>3.2 Related work</td>
<td>22</td>
</tr>
<tr>
<td>3.3 Overview</td>
<td>24</td>
</tr>
<tr>
<td>3.4 Structural Scene Representation</td>
<td>25</td>
</tr>
<tr>
<td>3.4.1 Hierarchical scene structures</td>
<td>25</td>
</tr>
<tr>
<td>3.4.2 Scene object representation</td>
<td>27</td>
</tr>
<tr>
<td>3.5 Recursive Model of Indoor Scenes</td>
<td>29</td>
</tr>
<tr>
<td>3.6 Results, evaluation, and applications</td>
<td>33</td>
</tr>
<tr>
<td>3.6.1 Scene generation results</td>
<td>34</td>
</tr>
<tr>
<td>3.6.2 Comparisons</td>
<td>38</td>
</tr>
<tr>
<td>3.6.3 Validation of network design and scene encoding</td>
<td>42</td>
</tr>
<tr>
<td>3.6.4 Applications</td>
<td>45</td>
</tr>
<tr>
<td>3.7 Discussion, Limitations, and Future Work</td>
<td>47</td>
</tr>
<tr>
<td>3.8 Appendix</td>
<td>49</td>
</tr>
<tr>
<td>4 Neural Graph Matching for Structural Layout Similarity</td>
<td>53</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>53</td>
</tr>
<tr>
<td>4.2 Related Work</td>
<td>55</td>
</tr>
<tr>
<td>4.3 Method</td>
<td>57</td>
</tr>
<tr>
<td>4.3.1 Layout Graphs</td>
<td>57</td>
</tr>
<tr>
<td>4.3.2 Graph Matching Network</td>
<td>58</td>
</tr>
<tr>
<td>4.3.3 Training</td>
<td>60</td>
</tr>
<tr>
<td>4.4 Datasets</td>
<td>61</td>
</tr>
<tr>
<td>4.5 Results and Evaluation</td>
<td>61</td>
</tr>
<tr>
<td>4.5.1 Baselines</td>
<td>61</td>
</tr>
<tr>
<td>4.5.2 Evaluation Metrics</td>
<td>62</td>
</tr>
<tr>
<td>4.5.3 Fully-connected vs. Adjacency Graphs</td>
<td>65</td>
</tr>
<tr>
<td>4.5.4 Ablation Studies on Structural Representation</td>
<td>66</td>
</tr>
<tr>
<td>4.5.5 Attention-based Layout Label Transfer</td>
<td>67</td>
</tr>
<tr>
<td>4.6 Conclusion, limitation, and future work</td>
<td>68</td>
</tr>
<tr>
<td>4.7 Appendix</td>
<td>68</td>
</tr>
<tr>
<td>5 Recovering Shape Interiors from Sparse Articulation Images</td>
<td>74</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>74</td>
</tr>
<tr>
<td>5.2 Related work</td>
<td>77</td>
</tr>
<tr>
<td>5.3 Overview</td>
<td>79</td>
</tr>
</tbody>
</table>
List of Tables

Table 3.1 Some statistics for the training dataset used in our experiments. The object categories included in the table cover the most frequently appearing objects in SUNCG. The remaining, infrequent, objects were removed. 34

Table 3.2 Performance of semantic scene segmentation using PointNet trained on different datasets. Our generated scenes can be used to augment SUNCG to boost performance. 47

Table 4.1 Precision scores for the top-k retrieved results obtained using different methods, on a set of randomly chosen UI and floorplan queries. The first set of five comparisons is for UI layouts, followed by floorplans. 63

Table 4.2 Overlap scores for checking the consistency of retrievals for a query and its top-1 retrieved result, over 50 such pairs. The first set of three rows are for UI layouts, followed by floorplans. 64

Table 4.3 Classification accuracy on test triplets obtained using IoU metric (IoU-based) and annotated by users (User-based). The first set of comparisons is for UI layouts, followed by floorplans. 65

Table 4.4 Precision@K scores for ablation studies on structural encoding of floorplan graphs. The setup for crowd-sourced relevance judgements via AMT is the same as in Table 4.1, on the same set of 100 randomly chosen queries. 67

Table 5.1 Quantitative evaluations of our motion-aware pose estimation network against PoseFromShape [156] and a variant of our method without the segmentation branch. 86

Table 5.2 Mean prediction errors on different motion parameters for RGB(D) image inputs from the test set: Raw network predictions vs. post-processing these raw predictions. $R$ refers to Revolute motions and $P$ for Prismatic. Only motion axis and origin are processed, since they are key to plausible motion realization in 3D. 86
Table 5.3  We compare to a RANSAC-based baseline for plane fitting on the predicted depth maps for interior plane predictions and recovery, and to shape completion; .......................... 87
List of Figures

Figure 3.1 We present a generative recursive neural network (RvNN) based on a variational autoencoder (VAE) to learn hierarchical scene structures, enabling us to easily generate plausible 3D indoor scenes in large quantities and varieties (see scenes of kitchen, bedroom, office, and living room generated). Using the trained RvNN-VAE, a novel 3D scene can be generated from a random vector drawn from a Gaussian distribution in a fraction of a second.

Figure 3.2 Architecture of our RvNN-VAE, which is trained to learn a generative model of indoor scene structures. Input to the network is a scene hierarchy composed of labeled OBBs enclosing 3D scene objects. The boxes are recursively grouped and codified by a set of encoders, resulting in a root code. The root code is approximated by a Gaussian distribution, from which a random vector is drawn and fed to the decoder. The recursive decoder produces an output hierarchy to minimize a reconstruction+VAE loss.

Figure 3.3 Overall pipeline of our scene generation. The decoder of our trained RvNN-VAE turns a randomly sampled code from the learned distribution into a plausible indoor scene hierarchy composed of OBBs with semantic labels. The labeled OBBs are used to retrieve 3D objects to form the final 3D scene.

Figure 3.4 A hierarchical vectorized encoding for an indoor scene. Leaf vectors record object sizes and labels; internal nodes contain positional information of the child nodes relative to that of the first child node.

Figure 3.5 A bedroom from the training set and the corresponding scene hierarchy. The root node always has a floor and four wall nodes as its children. Each wall is merged with its associated object clusters, which are organized into sub-trees based on finer-grained object-object relations.
Figure 3.6 Defining relative positions between a desk and a chair using one rotation angle and two offsets between closest edges from the two OBBs.

Figure 3.7 Along the horizontal direction, there are four cases of closest edge pairs from two axis-aligned boxes (blue and orange). The corresponding offset values are indicated by horizontal arrows.

Figure 3.8 Autoencoder training setup. The r-D vectors are the relative position vector between the objects as explained in each Encoder/Decoder module. Ellipsis dots indicate that the code could be either the output of BoxEnc, SuppEnc, Co-ocEnc or SurrEnc, or the inputs to BoxDec, SuppDec, Co-ocDec or SurrDec. The encoding process for objects in a scene is shown on the bottom right.

Figure 3.9 A scene generated by our method and its corresponding decoded hierarchy, which defines the scene structure.

Figure 3.10 Top three closest scenes from the training set, based on graph kernel, showing diversity of scenes from the training set.

Figure 3.11 Bedrooms generated by our method (a), in comparison to (b) closest scene from the training set, to show novelty, and to (c) closest scene from among 1,000 generated results, to show diversity. Different rows show generated scenes at varying complexity, i.e., object counts.

Figure 3.12 Several living room, kitchen, and office scenes generated by our method. More results can be found in the Appendix (Section 3.8.

Figure 3.13 Similarity plots between object co-occurrences in the training set vs. in the RvNN-generated scenes. Left: bedrooms; right: living rooms. Higher levels of similarity correspond to warmer colors. Note that we only plot similarities for pairs of categories with sufficiently high levels of co-occurrences; category pairs that rarely co-occur are discarded.

Figure 3.14 Comparison of scene generation with [65].

Figure 3.15 Statistics on plausible scene selection from the perceptual studies: The participants are asked to select the most plausible scene for each triplet, consisting of bedroom scenes from the training set, our results and the results from [65]. The plots show the percentage of rated plausible scenes for each method.

Figure 3.16 Results of plausibility rating by participants for bedrooms: The participants are asked to rate a scene based on their plausibility on a scale of 1 to 5 (5 being the most plausible). The values are the average scores for scenes from the training set, our results and the results by [65].
Figure 3.17  Statistics on plausible scene selection from the pairwise perceptual studies: The participants are asked to select the most plausible scene in a pair, consisting of scenes from bedrooms, living and offices, presented in a random order. The plots show the percentage of rated plausible scenes for synthesized scenes from the two methods against ours.

Figure 3.18  Typical bedroom scenes generated by our RvNN-VAE with various options involving wall and root encoders and decoders. (a) Without wall or root encoders and decoders. (b) With wall encoders and decoders, but not for root. (c) With both types of encoders and decoders.

Figure 3.19  Typical bedrooms generated using our method, but with alternate encodings of object position.

Figure 3.20  Typical bedroom scenes generated by our method when semantic labels are not included in object encodings in the leaf vectors, where 3D objects were retrieved based only on geometric information about the OBBs.

Figure 3.21  3D scene generation from 2D box layouts. Our trained RvNN-VAE can convert a 2D box layout representing the top-view of a scene (a) into a root code and then decode it into a 3D scene structure that closely resembles the 2D layout. (b) shows scenes decoded from the mean of the Gaussian of the latent vector and (c) shows scenes generated from latent vectors randomly sampled from the Gaussian.

Figure 3.22  Hierarchy-guided scene editing. Given a generated scene and its hierarchy (a), user can select a subtree (colored in green or blue) corresponding to a sub-scene (objects colored similarly). The subtree (sub-scene) can be replaced by another from other generated hierarchies, resulting in an edited scene (b).

Figure 3.23  (a) Generated living rooms

Figure 3.24  (b) Generated offices

Figure 3.25  (c) Generated kitchens

Figure 3.26  More results of other categories. Top two rows: generated living rooms; middle two rows: generated offices; last two rows: generated kitchens.

Figure 3.27  The object co-occurrence in the training scenes (top row) and our generated scenes (bottom row). From left to right are the plots of bedroom, living room, office, kitchen. Higher probability values correspond to warmer colors. Note that we only plot the co-occurrence between objects belonging to the frequent classes.
Figure 3.28 Similarity plots between object co-occurrence in the training set vs. in the RvNN-generated scenes. purpleLeft to right: bedroom, living room, office and kitchen. .......................... 51

Figure 3.29 The relative position distribution. The relative positions are obtained from: (a) indoor scenes from the training set; generated scenes each using (b) our relative position format, (c) absolute position format and (d) box transforms, respectively. The first row shows the center points of stands (blue dots) and beds (red rectangles), under the local frames of the beds. The second row shows the center points of chairs (blue dots) and desks(red rectangles), under the local frames of the desks. The arrows show the front orientations of beds and desks. Best viewed in color on Adobe Reader. .......................... 52

Figure 4.1 LayoutGMN learns a structural layout similarity metric between floorplans and other 2D layouts, through attention-based neural graph matching. The learned attention weights (numbers shown in the boxes) can be used to match the structural elements. .......................... 54

Figure 4.2 Structure matching in LayoutGMN “neutralizes” IoU feedback. In each example (left: floorplan; right: UI design), a training sample \( N \) labeled as “Negative” by IoU is more structurally similar to the anchor \( A \) than \( P \), a “Positive” sample. With structure matching, our network predicts a smaller \( A \)-to-\( N \) distance than \( A \)-to-\( P \) distance in each case, which contradicts IoU. .......................... 55

Figure 4.3 Given an input floorplan image with room segmentations in (a), we abstract each room into a bounding box and obtain layout features from the constituent semantic elements, as shown in (b). These features form the initial node and edge features (Section 4.3.1) of the corresponding layout graph shown in (c). .......................... 57

Figure 4.4 LayoutGMN takes two layout graphs as input, performs intra-graph message passing (Eq. 4.2), along with cross-graph information exchange (Eq. 4.3) via an attention mechanism (Eq. 4.5, also visualized in Figure 5.1) to update node features, from which final graph embeddings are obtained (Eq. 4.7). .......................... 58
Figure 4.5 Given a triplet of graphs $G_a$, $G_p$ and $G_n$ corresponding to the anchor, positive and negative examples respectively, the anchor graph paired with each of other two graphs is passed through a Graph Matching Network (Fig 4.4) to get two 1024-D embeddings. Note that the anchor graph has different contextual embeddings $h_{Ga}$ and $h'_{Ga}$. LayoutGMN is trained using the margin loss (margin=5) on the $L_2$ distances of the two paired embeddings. .......... 60

Figure 4.6 Top-5 retrieved results for an input query based on IoU metric, GCN-CNN_Triplet [93] and LayoutGMN. We observe that the ranked results returned by LayoutGMN are closer to the input query than the other two methods, although it was trained on triplets computed using the IoU metric. Attention weights for understanding structural correspondence in LayoutGMN are shown in Figure 5.1 and also provided in the Appendix (Section 4.7). UI and floorplan IDs from the RICO dataset [20] and RPLAN dataset [151], respectively, are indicated on top of each result. More results can be found in the Appendix(Section 4.7). .................. 62

Figure 4.7 Retrieval results for the bottom-left query in Fig 4.6, when adjacency graphs are used. We observe, on most of the queries, that the performance of LayoutGMN improves, but degrades in the case of GCN-CNN [93] on floorplan data. .................. 66

Figure 4.8 Element-level label transfer results from a source image $I_1$ to a target image $I_2$, using a pretrained LayoutGMN vs. maximum pixel-overlap matching. LayoutGMN predicts correct labels via attention-based element matching. .................. 67

Figure 4.9 Given a query (on the left) and its retrieved result (on the right), we show attention weights in different layers of message propagation, leading to element correspondences, from which structural similarity is driven and partly established. Layer-0 to Layer-5 show learned attention weights in different layers of propagation. For brevity, we only show the largest weights. .................. 69

Figure 4.10 Additional retrieved results on floorplan queries, using adjacency graphs, and fully-connected graphs, using both, GCN-CNN [93] (left column), and LayoutGMN (right column). Note that all the quantitative results shown in the main paper are based on fully-connected graphs, following the design choice of [93]. .................. 71
Figure 4.11 Additional retrieved results on UI layout queries, using adjacency graphs, and fully-connected graphs, using both, GCN-CNN [93] (left column), and LayoutGMN (right column). Note that all the quantitative results shown in the main paper are based on fully-connected graphs, following the design choice of [93].

Figure 4.12 Retrieved results for a given query and its top ranked retrieval, using GCN-CNN [93] (left column) and LayoutGMN (right column). In every set of paired results (row-wise), the first row represents a query $q$ and its top-5 retrievals. In the second row, the query is the top-1 result of query $q$ in the first row, denoted by $q^{\text{top-1}}$. Its top-5 retrievals shown against it.

Figure 5.1 RoSI is a learning framework trained to recover the interior of a 3D model from articulation images which capture its various part articulations. We show results produced from three input RGB images (a) for a 3D cabinet provided in its rest (unarticulated) state; see (b)-left. As illustrated in (b)-right by an exploded view, the input 3D model has no interiors. Our method not only infers the interior planes, shown in (c) in solid colors, but can also reproduce, and extrapolate (e.g., door opening), the captured motions on the 3D model (d).

Figure 5.2 Method pipeline. Separate networks are trained for per-view camera pose estimation, depth estimation, and articulation detection in the first phase. The predicted poses and depths are fed into the second phase for per-view interior plane recovery, followed by multi-view fusion. The rest-state (i.e., unarticulated) 3D model is only used as input to the pose estimation network and optionally, for articulation detection.

Figure 5.3 Our motion-aware architecture for pose estimation. Softmax and sigmoid functions are omitted in the figure for brevity.

Figure 5.4 Qualitative results on pose estimation. We show results from two viewpoints in each example. Left to right: rest-state 3D model, 3D bounding box overlaid on image using PoseFromShape, 3D bounding box using our method, our unarticulated area mask, and the GT mask. Green and blue boxes denote the GT and the prediction, respectively. Images in column (b) and (c) are enlarged for visualization purpose.
Figure 5.5 Qualitative results on articulation detection from a single image, with the top row showing network predictions and the bottom row shows the ground truth (GT). The arrows indicate the axis of motion and the small circles at the arrow indicate motion origin. Blue color is for GT motions and green color indicates that the motion predictions are in the local neighborhood of the ground truth.

Figure 5.6 Results on interior plane detection where each predicted plane is visualized using a different color. The top row shows the input image, middle shows predictions, and bottom row is for the ground truth.

Figure 5.7 Hallucinating interior geometry via shape completion using the method presented in [183]. For each result, we show one reference articulation image and point clouds of both input (i.e., exterior shell) and output for shape completion.

Figure 5.8 RoSI results on untrained object category, table – two input images and corresponding recovered interior planes.

Figure 5.9 Plane detection results on real images from the ABO dataset.

Figure 5.10 Qualitative results on interior recovery from three multi-articulation images. The first three columns show the input, the next two columns show the recovered planes visible in the input images, overlayed on the input 3D shell (semi-transparent) corresponding to the input images. The next three columns show the articulated 3D models with interior structures, rendered from the same viewing angle as the input images. Articulations are realized in 3D based on part detection and motion predictions from individual images.

Figure 5.11 Left to right – RGB Art (Articulated RGB image), Art mask (articulated object mask), Ori image (Rest state object image), Ori mask (rest state object mask), Arti area mask (mask of articulated parts of the 3D object), Inte area mask (mask of the interior regions of the 3D model). Given a 3D model, we shoot rays to its surface, both in the rest state and the articulated state. This is done to identify interior regions by finding newly intersecting faces compared to intersecting faces when rays hit the rest state 3D model. This setting allows us to get all of the metadata needed for all the modules presented in Figure 2 of the main paper.

Figure 5.12 Left to right – Input RGB image, GT articulation masks and motion parameters, unarticulated GT depth map, GT interior depth map, predicted interior depth map and depth map difference.
Figure 5.13  Left to right – Input RGB image, GT articulation masks and motion parameters, unarticulated GT depth map, GT interior depth map, predicted interior depth map and depth map difference.
Chapter 1

Introduction

“I like the new condo but I wish we had some say in designing the floorplan, or at least, choose among a few that resembled our previous unit. The property management should acquire technology that can show a bunch of similar floorplans based on the buyers’ preferences. Anyway, I am glad we bought that specific sofa set I was vouching for. It fits perfectly well in the living room of our new house” said the wife on the way back to their new place from a popular furniture store. On arrival, as the husband-daughter duo was busy unpacking and assembling different furniture, the wife said, "This is where we shall put our brand-new TV and the TV stand, which means the sofa set goes against the opposite wall, allowing us to get the sea view sitting on the sofa, while also providing enough room for movement".

Before the husband could reply, his three-year-old interrupted the conversation, "Daddy, the TV stand does not have shelves to store my toys". Hearing this, the wife asked the husband, "The store’s virtual reality app showed that the TV stand was vertically split, with a door on the left and shelves on the right. Didn’t it?". "Yes, that was the whole point of buying this particular TV stand since it gave ample storage space, or so we thought. It appears that the system is just for representative purposes", answered the husband. A visibly disappointed tech-savvy wife added, "I wish they had the technology that could automatically recommend different space designs for different room types, while also allowing interactions with the placed furniture items to understand how they function and how their interiors would look". The kiddo gently singled out two more side tables that lacked storage space. "Would you mind returning these and getting new ones that actually come with shelves or drawers and storage space? Make sure to physically check before you buy this time". The husband had no choice but to oblige.

1.1 Motivation

All of us, at some point, go through a similar situation of having to design our interior spaces, starting from a floorplan/blueprint, giving due consideration to both the design aesthetics (which kind of sofa and armchairs would go well with the wall carvings and the
ceiling arches), the intended usage of the rooms (a living room is typically intended for putting up a casual atmosphere, ex. a sofa-TV setting for watching movies) and the kind of interactions and functionalities afforded by potential objects comprising the indoor environment. A considerable amount of time, effort, and thought is put into designing home interiors, especially when starting from an empty space. This abstract premise should provide enough motivation to model 3D indoor scenes and articulated objects, from a functional usage perspective. Below, I expand this premise with details about modeling necessities at both the scene layout level and the object level and discuss them in the learning paradigm.

1.1.1 Why model 3D indoor scenes?

It is natural to iterate furniture placements many times in our heads before physically realizing a scene layout in the real world. Moreover, when a scene is being designed in real life, multiple re-arrangements may be performed until the layout looks satisfactory to one’s needs and functional usage. This is because there is no specific set of objects and/or furniture items to choose from, and no unique way of populating them in an indoor scene – it all depends on the available floorplan and the intended purpose of the indoor space, which in turn, dictates the purchase of a specific set of objects/furniture informed by the functionalities afforded by them.

Undeniably, this is a creative design task leveraging much of the human intelligence centered around perception (for what purpose would I need this object in this room and, overall, does the object placement serve me well for this room?) and interaction (how do I use these object for their intended functions?), something that can be aided by computer simulations. A necessary, ground-up, starting point for these computer simulations would be the availability of a large collection of 3D furniture models (such as ModelNet [152], ObjectNet [155], ShapeNet [9]), which can be utilized to simulate indoor environments with associated levels of granularity. Of course, one would prefer to spend less time doing such design tasks, which means manual modeling is not a viable path since it is laborious, time-consuming, and requires proficient modeling skills. The development of computational tools to better support such tasks is needed to assist users in realizing their designs, both at the object level and the scene layout level. Besides, such tools find wide industrial applicability ranging from augmented & virtual reality (AR/VR) applications, video games, and robotics, to creating virtual runs for artificial intelligent (AI) agents that live and interact in those environments. This also serves the graphics community well and helps advance the field of computer graphics. After all, one of the core research areas in computer graphics is that of content creation and understanding, especially in the 3D domain.

1.1.2 Why model articulated objects?

The usability of indoor scenes in real life is, in parts, determined by the objects that comprise them. Oftentimes, these objects are interactable, which allows them to be understood in
the way they function and their potential employability within indoor environments. Object interactions are typically carried out in the form of articulations on their parts, which are tied to the object geometry. And different parts of an object can undergo different kinds of motions (typically one of the two-prismatic and revolute). As well, when an object undergoes part articulations, its interiors are inevitably revealed.

As a first step in building agents, both virtual and real, that can interact with objects in the space they live in (in our case, we are looking specifically at indoor environments), it becomes necessary to first understand the objects that can potentially be interacted with (this is under the assumption that interactable objects are already localized in the agent’s vision system, in a part motion-unaware manner). When I say “understand the objects”, I am referring to their ability to decode what parts of the objects undergo movements/articulations, the way they undergo those motions, if the object is already in an articulated state, and if yes, observe how are the object interiors structured, and based on these, predicting articulations on similar objects (both intra- and inter-class objects; ex. a desk and a cabinet both have drawers) and how the interiors may potentially look like when a new object model from the same category is encountered in different settings. Answering these questions allows us to gain fine-grained knowledge of the functional utility of objects, which in turn, can guide object selections for different rooms.

1.1.3 Why learn to model them?

Modeling of 3D data (also applies to visual data in general) can be broadly categorized into two kinds: model-driven approaches and data-driven approaches. Model-driven approaches are those where the universe is more or less known apriori. With this knowledge, algorithms can be developed that follow a limited set of rules exhibited by the data samples in the universe. What stops this approach from finding general acceptance is that the universe on which these models are based is quite small, with little-to-no diversity, which means that these models can not be used as general-purpose computational tools. In the context of creative tasks such as designing indoor environments or 3D shapes, one example each of this approach includes works such as [95, 158].

On the other hand, data-driven approaches are not explicitly modeled based on a set of rules. They are designed to extract patterns directly from the data and use the information gained from observed data to make predictions on unseen data. The earliest and prominent works in this direction for 3D shape and scene synthesis include [13, 12, 27, 159, 29, 119, 91]. However, the learning approaches presented in the above works are based on smaller datasets and incorporate task-specific scene optimization algorithms that are difficult to converge and computationally expensive. Nevertheless, they provide inspiration for designing neural network-based models which can learn all sorts of patterns from the data, including those we fail to see or do not care about (a.k.a the data nuisance [71]), from large amounts of data. These neural nets are trained based on gradient backpropagation technique [118]
where sophisticated optimization algorithms (such as ADAM [69], AdaGrad [23], AdaDelta [174], RMSProp [43] etc.) are incorporated that allow for better convergence. And therefore the shift towards neural modeling.

To understand the above distinction better, let us consider an example task of placing an office desk in an empty room and using its drawers for storage purposes. This task involves reasoning and negotiating about fundamental questions, such as “what makes an object a desk?”, “what kind of desk to choose?”, “where to position the desk and how to orient it?”, “which parts of the desk are interactable?”, “how do they move?”, “is it possible to predict similar movements on other interactable parts of the desk?”, and many more.

Model-based approaches would be designed using a universe model that dictates the placement of desks using observed/laid-down rules. One of the rules could be that the desk should be placed against a wall with a window. If there is no window, then, the model will select another set of rules to place the desk; say, in the corner of the room. What if there already exists an object in both the places pointed above? In that case, another carefully crafted rule has to be made available. In other words, this simple problem setting shall elicit a set of rules that need to be drafted to handle different use cases. It can be seen how problematic such approaches can be when considering diverse settings (in the above case, the simplest diversity case can be different floor sizes and wall layouts). Put differently, it is rarely tractable, let alone desired, to explicitly model these questions in all kinds of environmental settings.

Being able to build algorithms that can extract and reuse relevant priors from a database of such real-world scenes and object interactions to solve the above problem (and other such problems) demonstrates the robustness and generalization ability of the developed algorithms. This “extract and reuse' is nothing but learning, something that is at the core of data-driven approaches. In essence, learning from a database of indoor scenes and interactable shapes allows for latent modeling of complex patterns within them (see Section 1.2 for the kind of complex patterns being referred to here), which otherwise can not be delineated, if at all, by a set of simple rules and/or algorithms.

The pursuit of the learning goal has been revived in the past decade with a remarkable development in computing technology, including but not limited to, hardware, compute, and machine learning algorithms. This thesis presents three works that have followed the spirit of their times in terms of technical approaches. The specific problems addressed in this thesis and their associated challenges are described below.
1.1.4 Specific problems addressed

Modeling is a general term that encompasses a wide range of problem settings. To narrow down the scope of this word, three specific problems are addressed in this thesis. All of these works center around the design of 3D indoor scenes, with advanced algorithmic development and evaluation in mind, going from the scene layout level to the object level, where functionality plays a key role in both cases.

First, we tackle the problem of automatically synthesizing 3D indoor scenes using neural networks by learning object occurrence patterns that conform to the functional usage of indoor space. To do this, we employ an end-to-end generative neural network that is trained on a dataset of synthetically created scenes. The underlying assumption here is that the synthetic scenes follow common functional rules on the layout level since they are manually created. This trained network is then used to produce different variations of indoor environments for individual room types. This is addressed keeping in mind the need for a design assist tool (idea-wise) for interior designers.

There is, however, a clear lack of principled ways to compare and evaluate the generated scenes. This leads to the second problem being addressed in the thesis. As a first step in this direction, we present a neural graph matching approach for structural layout similarity of abstractions of 3D indoor scenes. In particular, 2D floorplans are taken as prime examples. This caters to the first point made by the wife at the start of this chapter.

And finally, we move our focus from the scene layout level to the object level, again in the context of functionality. We introduce and address the problem of recovering interior structures of articulated 3D objects based on their 2D projections as input. The solution to this problem is a multi-stage learning framework where each stage leverages the representational power of deep neural networks. The purpose of addressing this problem is to supplement virtual agents in indoor environments with an interaction-based understanding, and also guide the interior design process based on object functionality.

1.2 Challenges

This section will describe the challenges involved in the three specific problems mentioned above. To be specific, we will be looking at these challenges from a learning perspective, with deep neural networks as our computational modeling tools.

1.2.1 Neural Generation of 3D Indoor Scenes

The basic building blocks of an indoor scene are 3D objects whose semantic composition and spatial arrangements follow clear patterns but still exhibit rich structural variations even within the same scene category. This can be imagined from the example of populating a living room starting from an empty space. Despite exhibiting clear object patterns, indoor
scenes offer a multitude of challenges, both in terms of representation and in the context of object relations. They are:

1. Indoor scenes are highly structured models. They could be represented as a volume or using multiview images or point clouds, which can then be processed using 3D or 2D convolutional networks or using point-based neural networks, respectively. However, such operations are oblivious to the underlying structures in the data that often play an essential role in scene understanding.

2. Relations between objects in a scene are much looser. Unlike in 3D shapes, the objects are almost never physically connected. As well, due to perturbations arising from human usage, symmetries (e.g., the rotational symmetry of dining chairs around a round table) are often not strictly adhered to.

3. Object-object relations in a scene are often more about semantics or functionality than geometric proximity or fit, with the latter being more prominent for part-part relations in 3D object geometry. For example, a TV and a sofa are related since they together serve the function of “watching TV”, but the two objects can be far apart in a scene. Unlike symmetry and proximity, relations of this kind are much more difficult, if not impossible, to infer based purely on geometric analysis or by analyzing a given scene alone.

4. Despite the rich intraclass structural variations that exist with manmade objects, e.g., the variety of chairs or airplanes, these objects can still be well aligned, at least globally. Such an alignment offers predictability in the positioning of object parts, which often plays an essential role in facilitating the learning of models or templates for manmade shapes [52, 68, 79]. In contrast, it is significantly more difficult to align scene layouts, even if they are all of bedrooms or living rooms.

1.2.2 Learning Structural Similarity for 2D Floorplans

Just as objects are to 3D scenes, the basic building blocks of a floorplan are the individual rooms, whose semantic composition and spatial arrangements can exhibit rich diversity. In other words, the subdivisions that compose floorplans can present significant structural variations. In order to develop an algorithm that can compare two floorplans, the following challenges need to be addressed:

1. The number of rooms in the two floorplans can be quite different. In addition, rooms of the same type can have different areas leading to significantly different floorplan perimeters. It is therefore not straightforward to establish a room-to-room correspondence, if any, between the two floorplans.
2. The key challenge in comparing two floorplans is that it is not purely a task of visual comparison — it depends critically on inference and reasoning about structures, which are expressed by the semantics and organizational arrangements of the elements or subdivisions which compose the floorplans, as explained above.

3. This then leads to the next question – how to represent floorplans for this comparison (and later retrieval) task?

### 1.2.3 Learning to Recover 3D Shape Interiors from Few Articulation Images

In the context of 3D shapes that are interactable or exhibit explicit articulations, we aim to model their interior structures based on multi-view image inputs. As well, we also aim to realize the part motions in 3D on the object, based on estimating articulation parameters from instances of 2D articulation images (the same set of multi-view images as before). However, 3D geometry inference from sparse RGB images is a severely ill-posed problem, even when the images are static across different views. The specific challenges encountered when tackling this problem are:

1. A unique challenge in our work comes from the input images, which capture an object undergoing different part articulations, with no constraints on the number of articulated parts in each image.

2. In addition, the input images may exhibit significant diversity when they were captured from varying camera poses, both in terms of viewing distance and angle.

3. These, along with the lack of image overlap due to view sparsity, would significantly compromise conventional multi-view 3D reconstruction approaches built on image correspondence.

4. An additional implied difficulty arises when estimating camera poses, which are unknown but necessary to place the per-view predictions into a common coordinate frame, for structure integration and motion realization.

### 1.3 Overview of the thesis

In this section, we provide an overview of the three works we developed to address the above challenges, in the context of learning to model 3D indoor scenes and articulated objects.

#### 1.3.1 Neural Generation of 3D Indoor Scenes

To address the first challenge described in Section 1.2.1, we present a generative neural network which enables us to generate plausible 3D indoor scenes in large quantities and varieties, easily and highly efficiently. Our key observation is that indoor scene structures are
inherently hierarchical. Hence, our network is not convolutional; it is a recursive neural network or RvNN [125]. Using a dataset of annotated scene hierarchies, we train a variational recursive autoencoder [70], or RvNN-VAE, which performs scene object grouping during its encoding phase and scene generation during decoding. Specifically, a set of encoders are recursively applied to group 3D objects based on support, surround, and co-occurrence relations in a scene, encoding information about objects’ spatial properties, semantics, and their relative positioning with respect to other objects in the hierarchy. By training a variational autoencoder (VAE), the resulting fixed-length codes roughly follow a Gaussian distribution. A novel 3D scene can be generated hierarchically by the decoder from a randomly sampled code from the learned distribution. We coin our method GRAINS, for Generative Recursive Autoencoders for INdoor Scenes. We demonstrate the capability of GRAINS to generate plausible and diverse 3D indoor scenes and compare with existing methods for 3D scene synthesis. We show applications of GRAINS including 3D scene modeling from 2D layouts, scene editing, and semantic scene segmentation via PointNet [112] whose performance is boosted by the large quantity and variety of 3D scenes generated by our method.

1.3.2 Neural Graph Matching for Structural Similarity of 2D Floorplans

Finally, to address the challenge described in Section 1.2.2, we present a deep neural network to predict structural similarity between 2D layouts by leveraging Graph Matching Networks (GMN) [82]. Our network, coined LayoutGMN, learns the layout metric via neural graph matching, using an attention-based GMN designed under a triplet network setting. To train our network, we utilize weak labels obtained by pixel-wise Intersection-over-Union (IoUs) to define the triplet loss. Importantly, LayoutGMN is built with a structural bias which can effectively compensate for the lack of structure awareness in IoUs. We demonstrate this on two prominent forms of layouts, viz., floorplans and UI designs, via retrieval experiments on large-scale datasets. In particular, retrieval results by our network better match human judgement of structural layout similarity compared to both IoUs and other baselines including a state-of-the-art method based on graph neural networks [93] and image convolution. In addition, LayoutGMN is the first deep model to offer both metric learning of structural layout similarity and structural matching between layout elements.

1.3.3 Recovering 3D Shape Interiors from Sparse Articulation Images

The dominant majority of 3D models that appear in gaming, VR/AR, and those we use to train geometric deep learning algorithms are incomplete, since they are modeled as surface meshes and missing their interior structures. To address the challenge described in Section 1.2.3, we present a learning framework to recover the shape interiors (RoSI) of existing 3D models with only their exteriors from multi-view and multi-articulation images. Given a set of RGB images that capture a target 3D object in different articulated poses, possibly from only few views, our method infers the interior planes that are observable in the input
images. Our neural architecture is trained in a category-agnostic manner and it consists of a motion-aware multi-view analysis phase including pose, depth, and motion estimations, followed by interior plane detection in images and 3D space, and finally multi-view plane fusion. In addition, our method also predicts part articulations and is able to realize and even extrapolate the captured motions on the target 3D object. We evaluate our method by quantitative and qualitative comparisons to baselines and alternative solutions, as well as testing on untrained object categories and real image inputs to assess its generalization capabilities.

1.4 Thesis contribution

To corroborate the narrative and the claims made thus far, this thesis bases its existence on the following three publications of mine:


This work presents the first hierarchical deep generative model for scene synthesis, and is one of the two first works employing neural networks for scene synthesis. The developed neural model generates an entire indoor scene in a single non-iterative forward pass, within fractions of a second (< 0.1 sec).


As a first step in developing a learned similarity metric for 3D indoor scenes, this work starts off in this direction by considering abstractions of 3D layouts, essentially exploring the 2D domain. It presents the first deeply learned metric that performs structural similarity of 2D layouts, ex., on floorplans. This is made possible due to attention-based graph matching. In doing so, our network reveals element-level correspondences, which are shown for this particular application for the first time.


This is the first work that introduces and addresses the novel problem of recovering 3D shape interiors based on multi-view, multi-articulation images. As a byproduct of our system, this work can also recover part motions in 3D on a part-segmented 3D object based on articulation RGB image input.
1.5 Thesis organization

This thesis is structured in the form of the following chapters:

Chapter 2. This chapter is aimed at providing relevant background on existing modeling tools in the context of this thesis. In doing so, the chapter provides a comprehensive summary of relevant works in the aforementioned areas (Section 5.3) and discusses how the contributions made by this thesis differ from them, and the gaps filled by these contributions.

Chapter 3. This chapter discusses neural generation of 3D indoor scenes from a database of synthetically designed scenes. Details about the data representation, network architecture catered for this generative task, the training mechanism, the evaluation scheme, and the potential applications served, are presented exhaustively.

Chapter 4. This chapter describes a learning framework for measuring the structural similarity of layouts in general, with floorplans as the prime examples since they represent abstractions of 3D scenes. It looks at how the presented approach is effective in learning better embeddings of floorplans compared to a closely related work through various ablations and evaluations.

Chapter 5. This chapter describes a multi-stage learning framework to recover 3D shape interiors from a set of sparse RGB articulation images. All the details, including the dataset, different neural networks employed, and the evaluation schemes are presented.

Chapter 6. As a collection of take-home messages, this chapter provides a summary of the key insights accumulated in this thesis. The dissertation concludes by discussing thought experiments, in the context of the presented works, that have the potential to open up feasible avenues for future research.
Chapter 2

Background

This chapter is aimed at providing a background of relevant topics centered around this thesis. This is accomplished by discussing notable related works in the areas of 3D indoor scene synthesis, articulation estimation and interior recovery of 3D articulated objects, and similarity metrics for comparing floorplans. For each of these tasks, we first discuss model-driven techniques, if any. After that, we review deep learning techniques employed in each of these tasks and discuss how our method (which will be presented in the subsequent chapters) is different from theirs.

2.1 Indoor 3D Scene Synthesis

**Probabilistic synthesis** Relying on probabilistic reasoning over scene exemplars forms the core of example-based scene synthesis approaches [96, 27, 58, 119, 179]. Notably, Fisher et al. [27] develop a Bayesian network for object co-occurrences and model object placements using a Gaussian mixture model (GMM). To synthesize new scenes from an example scene, object contextual placements are sampled from the learned GMM model. Most of the probabilistic scene synthesis algorithms follow this paradigm, but with variations on both heuristics and models capturing object co-occurrences and their relationships.

**Progressive synthesis** Models that synthesize a scene by sequential placements of an object or a set of objects (called sub-scene) are said to be progressive in nature. This is a reflection of how scenes evolve in the real world – based on human actions, which in turn, depend on the functionality of objects present in the scene. This forms the basis of most human-centric scene synthesis works such as [92, 119], discussed later below. In other words, progressive scene synthesis involves user input in some form, be it activity-driven or language-driven [91]. Such methods can even be made interactive offering more controllability, where the overall system is localized at every synthesis step.

First instance of such progressive synthesis via interactive modeling was demonstrated by
Merrell et al. [96]. They developed a modeling tool for furniture layout arrangement based on interior design guidelines. The design guidelines are encoded as terms in a probability density function and the suggested layouts are generated by conditional sampling of this function. Another related work called *Make It Home* [168] offers an interactive modeling tool to synthesize furniture layouts by optimizing a layout function which encodes spatial relationships between furniture objects. *ClutterPalette* [169] presents another interactive modeling tool that progressively populates a scene by suggesting a set of possible objects, the priors of which are learned from the data, when a user clicks on a particular region of the scene.

More recently, Chang et al. [10] and Ma et al. [91] use language commands to drive scene synthesis. In Ma et al. [91], language commands are parsed into scene graphs, which are used to retrieve subscenes from a scene database. The key idea leveraged by this method is that semantic scene graphs act as a bridge between language commands and scene arrangements, and as such, aligning the scene graph from one domain will allow to retrieve corresponding scenes. To account for the lack of an exact match, the system also allows to augment retrieved subscenes with additional objects based on the language context. At each step, a 3D scene is synthesized by coalescing the retrieved sub-scene, with augmented objects, into the current scene.

**Human-centric scene synthesis** Fisher et al. [29] present a method that can produce multiple plausible 3D scenes from an input RGBD scan. Here, plausibility means that the synthesized 3D scenes allow for the same functional activities as the captured environment. A scene template is estimated based on the input scan that captures likely human activities (as a probabilistic map) over the scene space. The core model, called the activity model, encodes object distribution with respect to human activities, and would guide the synthesis process based on predicted activities in the scene template.

In a slightly different setting, Savva et al. [119] capture human poses with object arrangements in the scene based on human activity. The underlying modeling is a probabilistic model. The functional relationships between humans and objects in the form of physical contacts and visual-attention linkages are represented using what they call Prototypical Interaction Graphs (in short, piGraphs). Joint probability distributions over human pose and object geometries are encoded in the PiGraphs and learned from data and in turn, the learned PiGraphs serve to guide the generation of interaction snapshots.

A concurrent work from Ma et al. [92] guides the scene generation process from human activity. In contrast to the above two methods, observations of human-object interactions in this work come from 2D images. That is, the action models are learned from annotated photographs in the Microsoft COCO dataset [84], which makes the problem challenging since such 2D images do not contain object/human-pose designation. The key idea of the work is to formulate transition probabilities to account for a transition in human activity.
An action graph is constructed whose nodes correspond to actions and edges encode transition probabilities. Synthesizing a new scene would correspond to sampling from the model capturing action graph priors.

**Deep Generative Models** Wang et al. [139] present a deep convolutional, autoregressive approach for 3D scene synthesis. A scene is represented as a multichannel top-view image where each channel encodes the mask of an object in the scene, in addition to depth information. An autoregressive neural network is then trained with such images (corresponding to 3D scenes) to output object placement priors as a 2D distribution map. To synthesize a new scene, objects are sequentially placed based on the learned placement priors.

Different from the above concurrent method, our work coined GRAINS presents a generative neural network for 3D scenes that can efficiently generate a large quantity and variety of indoor scenes. The key observation in our work is that indoor scenes are inherently hierarchical (so they represent 3D scenes as hierarchies), and use a recursive neural network (RvNN) architecture coupled with a VAE to model the space of scenes following the pipeline shown in Chapter 3. Using a dataset of annotated scene hierarchies, we train an RvNN-VAE, which performs scene object grouping during its encoding phase and scene generation during decoding. Specifically, a set of encoders is recursively applied to group 3D objects (represented as semantically oriented bounding boxes) in a scene, bottom up, and encodes information about the objects and their relations, where the resulting fixed-length codes roughly follow a Gaussian distribution. To generate a new scene, a random vector is sampled from the learned Gaussian and branched down through the RvNN decoder to obtain the scene hierarchy. Shape models are retrieved from a shape database based on the semantics and dimensions of leaf nodes in the generated hierarchy.

The below works came out after our work, GRAINS, was published. For completeness, we include them here.

Using another structural representation, Wang et al. [138] present an autoregressive graph generative model called PlanIT, for 3D scenes based on Graph Neural networks employing message passing convolutions. They represent a 3D scene as a graph with scene objects as nodes and their spatial or semantic relationships by edges of a graph. During training, the network learns relationship priors between different kinds of objects in a scene type (ex: bedrooms). To generate a new scene from an empty or a partially complete scene (or scene graph), the learned autoregressive model is used to obtain a scene graph, which is instantiated via an image-based reasoning module to generate a 3D scene corresponding to that scene graph.

Zhang et al. [177] present a generative model for indoor scenes based on a GAN, which
learns to map a normal distribution to the distribution of primary objects in indoor scenes. In this work, a 3D scene is represented as a matrix that encodes all the information about every object in a scene. A scene is encoded into a latent vector by a set of interleaved sparse and fully connected layers. The decoder, which mirrors the encoder, generates scene matrices. A discriminator is trained to classify whether the input to it is a real scene or not. In addition, an image-based discriminator is also used to differentiate between the top-view renderings of 3D scenes.

Very recently, [141, 106] developed conditional generative models for 3D scenes by making use of attention-based Transformer models [136]. The advantage of using Transformer models is that they alleviate the need for hand-crafting spatial relationships between objects, and instead, implicitly learn object relations through attention mechanism. Specifically, Wang et al. [141] condition the generation process on two kinds of inputs – room layout (including the position of doors and windows), and text descriptions. They represent indoor scenes as a sequence of object properties, converting the scene generation task to a sequence generation one. During training, an empty room (represented by the floor dimensions) or a text description (encoded using one of GloVe [110], ELMo [111] or BERT [22] techniques) is input to their model along with a sequential ordering of object categories. The transformer model learns to sequentially generate the properties of the next object in the predefined ordered set. During inference, given the type of user input (empty room or text description), the trained model sequentially outputs an ordered set of objects and inserts them into the existing scene.

On the other hand, Paschalidou et al. [106] reduce the problem of scene generation to that of generating an unordered set of objects, where meaningful object arrangements are obtained by sequentially placing objects in a permutation-invariant fashion. They represent a scene as an unordered set of objects where each object is encoded using its category, size, orientation (relative to the floor normal) and location. During training, given a training scene with M objects, they randomly permute them and keep the first T objects (here T=3). The network is tasked to predict the next object to be added to the scene given the subset of kept objects, and the floor layout feature. During inference, they start with an empty context embedding C and the floor representation of the room to be populated. From here, they autoregressively sample attribute values from the predicted distributions; Once a new object is generated, it is appended to the context C to be used in the next step of the generation process until the end symbol is generated. To transform the predicted labeled bounding boxes to 3D models, object retrieval from the dataset based on Euclidean distance of the bounding box dimensions is performed.

Another recent work from Yang et al. [164] developed a conditional volumetric generative model of indoor scenes using a GAN framework. They represent scenes as voxels, and take the room size as a conditional input to a GAN that is trained to map the distribution of
indoor scene to a normal distribution. The discriminator is trained on depth and semantic images of the volumetric scenes. To this end, they employ a differentiable renderer to render depth and semantic maps of generated volumetric scenes, which are used with the depth, semantic maps of scenes from the training database for learning the GAN discriminator. At generation time, given a room size $\phi$ and a latent vector $z_s$ randomly sampled from the latent space, the trained volumetric GAN can generate a semantic scene volume that stores both layout and rough shapes of the objects instances in the room. To obtain the final 3D scene, they extract object instances from the semantic scene volume and replace them with the CAD models retrieved (based on Chamfer Distance) from a 3D object database.

2.2 Learning Structural Similarity for 2D Floorplans

There have been structural similarity metrics proposed for 3D shapes (Shape Edit Distance, SHED, [73]) and 3D scenes (Graph Kernel [28] and Focal-centric Graph Kernel [157]). These works provide valuable cues for developing an effective structural metric for layout similarity. To learn structural similarity, it is inevitable that the representations of floorplans be strongly structural in nature, which essentially boils down to either graphs or hierarchies. We discuss existing works in this direction and differentiate our method from such works.

**Structural layout similarity** The work by Manandhar et al. [93] is the first to leverage GNNs to learn structural similarity of 2D graphical layouts, focusing on UI layouts with rectangular boundaries. They employ a GCN-CNN architecture on a graph of UI layout images, trained under a triplet network [45] setting. They obtain graph embeddings for the anchor, positive, and negative graphs independently.

In contrast, our work, LayoutGMN, learns the graph embeddings in a dependent manner. Through cross-graph information exchange, the embeddings are learned in the context of the anchor-positive (respectively, the anchor-negative) pair. This is a critical distinction to GCN-CNN [93], while both train their triplet networks using IoUs. However, since IoU does not involve structure matching, it is not a reliable measure of structural similarity, leading to labels that are considered “structurally incorrect”.

The below works came out after our work, LayoutGMN, was published. For completeness, we include them here.

[2] presents a spatio-structural similarity metric for comparing two UI layouts, that learns from an annotated layout represented as a tree. They refer to visual characteristics of a layout as “spatial” and relational patterns among different elements as “structural”. That is, they represent a layout as a hierarchy of elements where at each level, both CNN features and graph embeddings of constituent elements at that level are obtained, which are
recursively aggregated to capture layout attributes at different granularities. [160] learns the similarity between two document layouts, constrained by hierarchical representation, using a structure correspondence algorithm that finds optimal substructure correspondence. This is similar to the DocSim metric proposed in [108], except that DocSim is a heuristic approach that directly performs element-to-element correspondence, as opposed to learning substructure correspondence presented in [160].

2.3 Recovering 3D Shape Interiors and Part Motions from Articulation Images

**Articulation detection from images.** Similar in spirit to the task of 2D object detection, the task of detecting articulated parts of furniture models from images aims at predicting the 2D part bounding box and part mask, as well as inferring various motion-related parameters on the articulated part such as motion type, motion axis, motion origin and motion magnitude. This finds utility in being able to transfer predicted motions from one articulated product image to many similar-looking static 3D models, thus helping create articulated 3D assets. To achieve this, a natural solution is to extend standard 2D object detection architectures, such as MaskRCNN [41]. Recent works from [57] and [114] build upon the MaskRCNN architecture, where the former focuses on detecting *openable* parts in articulated images and the latter doing so on articulation videos. Note that they do not focus on predicting motion magnitude. In our work, we build upon the work of [57] and detect all four motion parameters, in addition to 2D part bounding boxes and segmentation masks. Recent work from [18] presents a framework to estimate motion parameters and segmentation of articulated part on a 3D object from a pair of articulated single image input and 3D mesh in rest state. The learning framework is based on an image encoder (ResNet-18 [132] and a shape encoder (PointNet [112]), whose embeddings are concatenated to jointly estimate part motion parameters and the per-point movement mask. A major drawback of this work is that it can only consider a single part articulation. Recently, [140] present an active learning (AL) framework, based on a transformer-based segmentation network that utilizes a masked-attention mechanism, for instance segmentation of interactable parts from RGB images of real indoor scenes. This work only performs 2D segmentation and does not estimate any motion parameters for articulated parts of an object visible in the image.

**Neural articulated 3D shape reconstruction.** Recently, there have been works that learn category-specific neural shape and appearance representations for articulated 3D shapes, e.g, NASAM [145] and A-SDF [100], where multi-view inference has been considered as an application. Another work, StrobeNet [175], reconstructs articulated 3D objects from multi-view and multi-articulation images. However, like the other works, StrobeNet also learns a category-level neural reconstruction model, i.e., it is trained per category and not dedicated
to recover 3D shape interiors. In contrast, we target the specific problem of interior plane recovery and we do so in a *category-agnostic* manner.

**Recovering object interiors** *Proactive scanning* [162] is the only classical 3D reconstruction approach geared towards hidden object interiors. In this setting, human users actively adjust the target scene, e.g., opening a cabinet door or drawer, while continuously scanning it to reveal occluded regions. Despite its potential, this 3D acquisition paradigm introduces many technical challenges involving scan registration, human occlusion, as well as motion tracking and recognition. Our work is a clear contrast to the Proactive Scanning approach – we do *not* scan the object interiors; it relies on supervised learning to recover shape interiors from articulation images, while limited to what is visible.

**Shape Completion as an Alternative.** Shape completion is a possible alternative for synthesizing interior geometries, when the input consists of an exterior shell. While there have been many neural shape completion methods, mostly operating on point clouds [172, 105, 53] and with recent attempts utilizing transformers [183, 154, 163, 170], we are not aware of any work that specifically targets the recovery of shape interiors.
Chapter 3

Generative Recursive Autoencoders for Indoor Scenes

3.1 Introduction

With the resurgence of virtual and augmented reality (VR/AR), robotics, surveillance, and smart homes, there has been an increasing demand for virtual models of 3D indoor environments.

At the same time, modern approaches to solving many scene analysis and modeling problems have been data-driven, resorting to machine learning. More training data, in the form of structured and annotated 3D indoor scenes, can directly boost the performance of learning-based methods. All in all, the era of “big data” for 3D indoor scenes is seemingly upon us. Recent works have shown that generative neural networks can be trained to synthesize images, speech, and 3D shapes. An obvious question is whether similar success can be achieved with 3D indoor scenes, enabling us to easily generate a large number of 3D scenes that are realistic and diverse.

Figure 3.1: We present a generative recursive neural network (RvNN) based on a variational autoencoder (VAE) to learn hierarchical scene structures, enabling us to easily generate plausible 3D indoor scenes in large quantities and varieties (see scenes of kitchen, bedroom, office, and living room generated). Using the trained RvNN-VAE, a novel 3D scene can be generated from a random vector drawn from a Gaussian distribution in a fraction of a second.
The basic building blocks of an indoor scene are 3D objects whose semantic or functional composition and spatial arrangements follow clear patterns, but still exhibit rich structural variations even within the same scene category (e.g., think of the layout varieties of kitchens or bedrooms). Highly structured models, including indoor scenes and many man-made objects, could be represented as a volume or using multi-view images and undergo conventional convolutionary operations. However, such operations are oblivious to the underlying structures in the data which often play an essential role in scene understanding. This may explain in part why deep convolutional neural networks (CNNs), which have been so successful in processing natural images, have not been widely adopted for the analysis or synthesis of 3D indoor scenes.

In this paper, we present a generative neural network which enables us to easily generate plausible 3D indoor scenes in large quantities and varieties; see Figure 5.1. Using our approach, a novel 3D scene can be generated from a random vector drawn from a Gaussian in a fraction of a second, following the pipeline shown in Figure 5.2. Our key observation is that indoor scene structures are inherently hierarchical. Hence, our network is not convolutional; it is a recursive neural network [125] or RvNN¹.

Using a dataset of annotated scene hierarchies, we train a variational recursive autoencoder, or RvNN-VAE, which performs scene object grouping during its encoding phase and scene generation during decoding, as shown in Figure 3.2. Specifically, a set of encoders are recursively applied to group 3D objects in a scene, bottom up, and encode information about the objects and their relations, where the resulting fixed-length codes roughly follow a Gaussian distribution. A new scene can then be generated top-down, i.e., hierarchically, by decoding from a randomly generated code.

Our approach is inspired by the recent work of Li et al. [79], coined GRASS, which develops a generative recursive autoencoder for learning shape structures. Specifically, they model the part structure of a 3D object using a hierarchical organization of assembly and symmetry groupings over the object parts and train an RvNN built on autoencoders to learn hierarchical grouping rules. Like GRASS, our neural network for indoor scenes, which we call GRAINS, is also an RvNN-based generative autoencoder. But the data representation, network architecture, and training mechanism all have to be altered in fundamental ways to meet the unique challenges arising from learning indoor scene structures and object-object relations.

On a superficial level, one may regard parts in a 3D object as the same as objects in a 3D scene. While part-part and object-object relations, for objects and scenes respectively, are both highly structured, there are several critical distinctions between them:

¹RvNNs are not to be confused with recurrent neural networks or RNNs.
Figure 3.2: Architecture of our RvNN-VAE, which is trained to learn a generative model of indoor scene structures. Input to the network is a scene hierarchy composed of labeled OBBs enclosing 3D scene objects. The boxes are recursively grouped and codified by a set of encoders, resulting in a root code. The root code is approximated by a Gaussian distribution, from which a random vector is drawn and fed to the decoder. The recursive decoder produces an output hierarchy to minimize a reconstruction+VAE loss.

Figure 3.3: Overall pipeline of our scene generation. The decoder of our trained RvNN-VAE turns a randomly sampled code from the learned distribution into a plausible indoor scene hierarchy composed of OBBs with semantic labels. The labeled OBBs are used to retrieve 3D objects to form the final 3D scene.

- Predominantly, the constituent parts of a 3D object are strictly governed by connectivity and symmetry [142]. In contrast, relations between objects in a scene are much looser. The objects are almost never physically connected. As well, due to perturbations arising from human usage, symmetries (e.g., the rotational symmetry of dining chairs around a round table) are often not strictly adhered to.

- Object-object relations in a scene are often more about semantics or functionality than geometric proximity or fit, with the latter being more prominent for part-part relations in 3D object geometry. For example, a TV and a sofa are related since they together serve the function of “watching TV”, but the two objects can be far apart in a scene. Unlike symmetry and proximity, relations of this kind are much more difficult,
if not impossible, to infer based purely on geometric analysis or by analyzing a given scene alone.

- Despite the rich intra-class structural variations that exist with man-made objects, e.g., the variety of chairs or airplanes, these objects can still be well aligned, at least globally. Such an alignment offers predictability in the positioning of object parts, which often plays an essential role in facilitating the learning of models or templates for man-made shapes [68, 52, 79]. In contrast, it is significantly more difficult to align scene layouts, even if they are all of bedrooms or living rooms.

To address the new challenges as a result of these differences, GRAINS differs from GRASS in several key aspects. First, we construct our RvNN for indoor scenes using three grouping operations (with their corresponding encoders and decoders as shown in Figure 3.2): support (e.g., desk supporting a computer), surround (e.g., nightstands surrounding a bed or dining chairs surrounding a table), and co-occurrence (e.g., between sofa and coffee table or between computer, keyboard, and mouse). In contrast, the GRASS RvNN for 3D shapes is defined by two grouping operations: symmetry and connectivity. Generally speaking, the term “object co-occurrence” would encompass both support and surround relations. However, in our paper, co-occurrences is a “catch-all” entity that covers all the geometrically loose and functionality- or action-oriented object-object relations that do not reflect physical support or surround.

Second, a proper grouping of 3D objects in a scene has to account for object co-occurrences, which are inherently tied to the semantics of the objects. Indeed, co-occurrence hinges mainly on what the objects are, e.g., a computer monitor is almost always associated with a keyboard. Thus, unlike GRASS, our scene RvNN must encode and decode both numerical (i.e., object positioning and spatial relations) and categorical (i.e., object semantics) data. To this end, we use labeled oriented bounding boxes (OBBs) to represent scene objects, whose semantic labels are recorded by one-hot vectors.

Finally, GRASS encodes absolute part positions, since the global alignment between 3D objects belonging to the same category leads to predictability of their part positions. However, in the absence of any sensible global scene alignment over our scene dataset, GRAINS must resort to relative object positioning to reveal the predictability of object placements in indoor scenes. In our work, we encode the relative position of an OBB based on offset values from the boundaries of a reference box, as well as alignment and orientation attributes relative to this reference. Under this setting, the room walls would serve as the initial reference objects (much like the ground plane for 3D objects) to set up the scene hierarchies, and subsequent reference boxes are determined on-the-fly for nested substructures.

Figure 3.2 shows the high-level architecture of our RvNN-VAE and Figure 5.2 outlines the process of scene generation. We demonstrate that GRAINS enables us to generate a large number of 3D indoor scenes that are plausible and diverse, easily and highly effi-
ciently. Plausibility tests are conducted via perceptual studies and comparisons are made to existing 3D scene synthesis methods. Various network design choices, e.g., semantic labels and relative positioning, are validated with experiments. Finally, we show applications of GRAINS including 3D scene modeling from 2D layouts, scene manipulation via hierarchy editing, and semantic scene segmentation via PointNet [112] whose performance is clearly boosted by the large quantity and variety of 3D scenes generated by our method.

3.2 Related work

In recent years, much success has been achieved on developing deep neural networks, in particular convolutional neural networks, for pattern recognition and discriminative analysis of visual data. Some recent works have also shown that generative neural networks can be trained to synthesize images [135], speech [134], and 3D shapes [150]. The work we present is an attempt at designing a generative neural network for 3D indoor scenes. As such, we mainly cover related works on modeling and synthesis of 3D shapes and 3D scenes.

**Indoor scene synthesis.** The modeling of indoor environments is an important aspect of 3D content creation. The increasing demand of 3D indoor scenes from AR/VR, movies, robotics, etc, calls for effective ways of automatic synthesis algorithms. Existing approaches to scene synthesis mostly focus on probabilistic modeling of object occurrence and placement. The technique of Fisher et al. [?] learns two probabilistic models for modeling sub-scenes (e.g. a table-top scene): (1) object occurrence, indicating which objects should be placed in the scene, and (2) layout optimization, indicating where to place the objects. Given an example scene, new variants can be synthesized based on the learned priors.

Graphical models are recently utilized to model global room layout, e.g., [65]. To ease the creation of guiding examples, Xu et al. [159] propose modeling 3D indoor scenes from 2D sketches, by leveraging a database of 3D scenes. Their method jointly optimizes for sketch-guided co-retrieval and co-placement of scene objects. Similar method is also used to synthesize indoor scenes from videos or RGB-D images [14, 65]. Some other works perform layout enhancement of a user-provided initial layout [168, 95]. Fu et al. [33] show how to populate a given room with objects with plausible layout. A recent work of Ma et al. [91] uses language as a prior to guide the scene synthesis process. Scene edits are performed by first parsing a natural language command from the user and transforming it into a semantic scene graph that is used to retrieve corresponding sub-scenes from a database. This retrieved sub-scene is then augmented by incorporating other objects that may be implied by the scene context. A new 3D scene is synthesized by aligning the augmented sub-scene with the user’s current scene.

Human activity is another strong prior for modeling scene layout. Based on an activity model trained from an annotated database of scenes and 3D objects, Fisher et al. [?] synthe-
size scenes to fit incomplete 3D scans of real scenes. Ma et al. [92] introduce action-driven evolution of 3D indoor scenes, by simulating how scenes are altered by human activities. The action model is learned with annotated photos, from which a sequence of actions is sampled to progressively evolve an initial clean 3D scene. Recently, human activity is integrated with And-Or graphs, forming a probabilistic grammar model for room layout synthesis [113].

In a concurrently developed work, Wang et al. [139] learn deep convolutional priors for indoor scene synthesis. Their method encodes scene composition and layout using multi-channel top-view depth images. A deep convolutional neural network is then trained with these image channels to output object placement priors as a 2D distribution map. Scene synthesis is performed via a sequential placement of objects, guided by the learned priors. The key differences between their and our scene synthesis frameworks include: 1) GRAINS produces 3D indoor scenes with object-on-furniture support and is not limited to just floor-supported furniture layouts; 2) we model and learn scene hierarchies rather than a flat object layout; 3) we adopt a structural representation of indoor scenes which explicitly encode spatial relations and contextual information.

Generative 3D modeling. We focus on generative models for creating discrete variations of 3D models at the part level. Earlier models are either based on hand-crafted shape grammars [101], or learned from a single or a few training examples [5, 130]. More recent models are usually constructed from shape correlation across a larger set of 3D models, e.g., feature point correspondence [68] or part correspondence [158]. Based on part correspondence, some works explored the learning of part-based Bayesian networks [12, 62] or deformable templates [26, 52], to encode both continuous and discrete variations. Such models have been successfully extended to scene modeling [??, 33].

Deep generative models for 3D modeling. Deep generative networks including Variational Auto-Encoders (VAE) [70] and generative adversarial nets (GAN) [38] have enabled effective modeling of high-dimensional data with a low-dimensional latent space. New samples can be drawn from the latent space and mapped back to the original data space with high quality. Limited by the available vector representations, however deep generative models for 3D shapes have thus far been focusing on the generation of objects or scenes in volumetric representation [152, 37, 150, 127]. A major drawback of such models is the generated volumes are structure-oblivious; There is no part information and thus topology (e.g., connection) and structure (e.g, symmetry) are not well-defined. To learn a structure-aware model, we need to look into neural models for structural representations such as graphs.

Generative neural models for structures. To learn feature representation for general structures, other than 1D or 2D grids, some works attempted to extend convolutional neural networks to graph data [42, 24, 103]. However, it seems hard to utilize these networks to
Figure 3.4: A hierarchical vectorized encoding for an indoor scene. Leaf vectors record object sizes and labels; internal nodes contain positional information of the child nodes relative to that of the first child node.

3.3 Overview

Our RvNN-VAE framework for generating 3D indoor scenes is trained on a large 3D scene dataset, where each scene is composed of a set of labeled objects which are represented by bounding boxes (OBBs). Once trained, the RvNN-VAE is used to generate new scenes through decoding a randomly sampled noise vector into a hierarchy of OBBs with object labels. The labeled OBBs are then replaced with 3D objects retrieved from a 3D shape database based on object category and model dimensions. Figure 3.2 shows the high-level architecture of the RvNN-VAE and Figure 5.2 illustrates the generative process enabled by the learned deep neural network.

Structural scene representation Given an input 3D indoor scene composed of a set of labeled objects, we organize the objects including walls and floor of a scene into a hierarchy, based on their spatial relations (Section 3.4). Objects are present at the leaf level in the hierarchies, while each internal node represents a group of objects under its subtree. Figure 3.4 shows an illustration of hierarchical structure for indoor scenes. The object labels and sizes are encoded in the leaf vectors, and the spatial placement information is encoded by relative positions between sibling nodes in the hierarchy.
**Recursive VAE** To learn the layout of 3D indoor scenes, we train a Variational Auto-Encoder (VAE) whose encoder maps an indoor scene or more specifically, its OBB hierarchy, into a fixed length root code in a bottom-up, recursive fashion and the decoder works inversely (Section 3.5). This is illustrated in Figure 3.2. During encoding, the box encoder is first applied to each of the leaf vectors, to map them into fixed length leaf codes. Then, the RvNN repeatedly invokes one of the four encoders and outputs a code until the code for the root node is generated. The decoder module performs an inverse process to reconstruct the hierarchy with the help of a node classifier that is trained simultaneously to predict the node type.

**Scene generation.** Once trained, the network learns a distribution of root codes that correspond to various scenes. Therefore, given a root code sampled from the distribution, the trained decoder module will decode the root code into a hierarchy of OBBs, as shown in Figure 5.2. The decoded hierarchy includes the relative positions between sibling nodes and the bounding boxes in the leaf nodes.

### 3.4 Structural Scene Representation

Indoor scenes are often characterized by their layouts and the classes of objects present in them, which we represent using labeled oriented bounding boxes (OBBs). For a generated scene to look realistic, it should follow some common object placement patterns within its subscenes. Object placement pattern involves both object classes and the relative positions between the associated objects. We make the key observation that such patterns are hierarchical, that is, they follow grouping rules at multiple levels of abstraction. Hence, we employ a hierarchical model to organize the objects in a scene and record their relative positions. Figure 3.4 shows one such illustration. Here, when merging several nodes into internal nodes, we take into account their OBBs and the relative positions.

In this section, we describe the details of our structural scene representation, where our key innovation is the design of relative position encoding for object placements (Section 3.4.2).

#### 3.4.1 Hierarchical scene structures

Given an indoor scene, we organize the constituent objects into a hierarchy, where the objects and object groups are represented by the leaf nodes and internal nodes, respectively. We observe that the common features of indoor scenes lie within their corresponding subscenes, that include the objects and their placement w.r.t each other. To preserve these features, it is essential to build the hierarchies based on these sub-scenes.
Pre-defining object relations  Our idea is to first merge the commonly perceived “relevant” objects into sub-scenes and group the sub-scenes into a complete scene. However, because of the large variation in the training data in terms of number of objects, their geometry and placement across all the indoor scenes (even among scenes within the same room type), there’s no consistent way of defining the sub-scenes. In spite of such a large variation, the relevant objects usually have consistent spatial relations among various scenes. This can be taken as a heuristic rule to construct the hierarchies for all the scenes. For example, a desk is likely to have a chair nearby; a nightstand is likely to be placed adjacent to a bed; the cabinets are placed against the walls, etc.

Therefore, we build our hierarchies based on the spatial relations. To categorize these spatial relations, we characterize object-object relations into three categories: support, surround and co-occur. The support relation is defined for objects where one object is placed on top of the other. A set of objects is said to form a surround relation if they have similar size and same label, and are located around a central object. The co-occurrence relation is used as a “catch-all” relation. If two objects are not involved in a support or a surround relation, they are considered using the co-occurrence relation. In our hierarchies, we treat the walls and the floor as special objects because they serve as the “reference” for the scene layout. In particular, we observe that the walls are responsible for the correct orientation of objects in a scene.

Building training scene hierarchies  To build a scene hierarchy, we first cluster the objects in the scene based on the closest walls they are placed against. Then, we build a subtree using the object relations in every cluster. The support relation has the highest priority, followed by the surround relation and then the co-occurrence relation. Figure 3.5 shows a bedroom with various object relations and the corresponding scene hierarchy. Leaf nodes correspond to objects and non-leaf nodes represent object groups with certain relations. We observe that the two pairs of nightstands and table lamps are first merged by the support relation independently, and then merged with the bed with the surround relation.

To highlight these features to the network, we place the children of each kind of internal-node in a particular order. To elaborate, for a support node, its second child (object or object group) is supported by the first child (object). For a surround node, its first child is the central object around which the other two objects (object groups) are placed. For a co-occurrence node, the children are sorted based on the sizes of their OBBs – the first child having the largest OBB. These orders enhance the common features in the dataset, making it easier for the network to learn.

Once the object clusters are available, we merge the walls and the associated clusters to form the “wall nodes”. Then the floor and all the “wall nodes” are merged to get the “root node”. Figure 3.5 shows the overall hierarchy for a bedroom. In the hierarchy, we have different types of nodes (leaf nodes, support nodes, co-occurrence nodes, surround
3.4.2 Scene object representation

We represent a 3D scene based on semantic and geometric information about the objects therein and object-object relations, in a scene hierarchy, as shown in Figure 3.4. Specifically, in each leaf node, we record information about a 3D object, including the geometries (e.g., length, height, etc.) of its 3D OBB, under a local frame, and the object’s semantic label, encoded as a one-hot vector whose dimension equals to the number of object categories for the corresponding scene type. In each internal node, we record the relative positioning between the OBBs of two sibling child nodes. It suffices to encode 2D layout information since object positioning along the height dimension can be defined by the support relation. Overall, the relative (2D) object positioning is encoded in a 28-dimensional vector of real numbers and bits, as we explain below.

Orientation and offsets  To encode the relative positioning of one box, referred to as the target box, with respect to a reference box, we first rotate the target box so that it is axis-aligned with the reference, where the rotation angle is stored in the 28-D vector. Then
we store two offset values, each representing the distance between two closest edges from the two boxes, one along the horizontal direction and one along the vertical direction, as shown in Figure 3.6. In Figure 3.7, we show that along the horizontal direction, there are 4 possible cases for the pair of closest edges. Hence, we can use a $4 \times 4 = 16$-bit “two-hot” indicator vector to identify which two pairs of edges from the two boxes would define the two offset values.

**Attachment and alignment** Certain object-object attachment or alignment relations are rather strict, e.g., all beds have at least one side leaning against a wall. However, such strict relations are difficult to enforce precisely using only the angle and offset values since these values are easily affected by noise and one can always expect variations in them from the training data. To allow precise attachments and alignments in the generated scenes, we opt to encode such relations explicitly using bit codes as a means of reinforcement beyond encoding object positions using offsets and angles.

Specifically, attachment is encoded by a 4-bit one-hot vector indicating whether and how the two boxes are attached. The first bit is used to indicate whether or not the two boxes are attached, the next two bits indicate whether the boxes are attached along any one of the two axes, and the last bit indicates if the objects are aligned along both the axes. Note that an attachment implies that the two boxes are axis-aligned and that two of their edges either overlap or lie along a line, e.g., the left edge of the target box overlaps with the right edge of the reference box or the right edges of the two boxes lie along a line without overlapping.

Alignment is encoded by a 5-bit one-hot vector to indicate whether the two boxes are oriented at an angle of $0^\circ$, $90^\circ$, $180^\circ$, or $270^\circ$ with respect to one another, or none of the
above. In the latter case, note that the rotation angle is already stored in the 28-D vector.

To summarize, the 28-D vector contains three real numbers (rotation angle and two offsets) and 25 binary indicators (16 bits for edge pair identification; 4 bits for attachment; 5 bits for alignment).

3.5 Recursive Model of Indoor Scenes

In this section, we describe our method to learn a representation of indoor scenes, of varying complexity, as fixed-dimensional codes. Our model is based on Recursive Autoencoders (RAE) for unlabeled binary trees, developed by Socher et al. [125]. RAEs were originally intended for parsing natural language sentences in a discriminative setting, trained on unlabeled parse trees. It consists of an encoder neural network that takes two $n$-dimensional inputs and produces a single $n$-D output, and a decoder network that recovers two $n$-D
vectors from a single n-D vector. Given a binary tree with n-D descriptors for leaves, the RAE is used to recursively compute descriptors for the internal nodes, ending with a root code. The root code can be inverted to recover the original tree using the decoder. With an additional classifier that helps reconstruct the original tree topology from a given root code, the decoding process does not require access to the input hierarchy.

In our scenario, the primary goal of the scene encoder is to encode the hierarchical organization of objects in a manner consistently reflecting common patterns found in various scenes. We extend the original RAE framework for the task of scene generation to accommodate non-binary nodes (the surround and root nodes) and multiple encoder-decoder pairs. In addition to the non-binary nodes, our hierarchies are built based on various object-object relations with different features, and therefore it is natural to use specific encoder and decoder pair for each relation.

**Autoencoder model:** Our recursive autoencoder comprises six distinct encoder/decoder pairs. A box encoder is used to convert the leaf vectors into codes. To handle the object-object relations, we have support, co-occurrence and surround encoders. To group a cluster with its corresponding wall, we devise a wall encoder. Finally, a root encoder merges the four wall nodes with the floor. Figure 3.8 show the inputs and outputs of all these encoder/decoder pairs.

Below, we describe the encoders and decoders in detail. Each encoder and decoder (except those for input/output boxes) is a multi-layer perceptron (MLP) [98, 147], defined as a neural network with a finite series of fully-connected layers. Each layer \( l \) processes the output \( x_{l-1} \) of the previous layer (or the input) using a weight matrix \( W^{(l)} \) and bias vector \( b^{(l)} \), to produce the output \( x_l \). Specifically, the function is:

\[
x_l = \tanh \left( W^{(l)} \cdot x_{l-1} + b^{(l)} \right).
\]

Below, we denote an MLP with weights \( W = \{W^{(1)}, W^{(2)}, \ldots \} \) and biases \( b = \{b^{(1)}, b^{(2)}, \ldots \} \) (aggregated over all layers), operating on input \( x \), as \( f_{W,b}(x) \). Each MLP in our model has one hidden layer.

**Box** The input to the recursive merging process is a collection of object bounding boxes plus the labels which are encoded as one-hot vectors. They need to be mapped to n-D vectors before they can be processed by different encoders. To this end, we employ an additional single-layer neural network BoxEnc, which maps the k-D leaf vectors of an object (concatenating object dimensions and the one hot vectors for the labels) to a n-D code, and BoxDec, which recovers the k-D leaf vectors from the n-D code. These networks are non-recursive, used simply to translate the input to the code representation at the beginning, and back again at the end. Each network’s parameters comprise a single weight matrix, and a single bias vector.
**Support** The encoder for the support relation, $\text{SuppEnc}$, is an MLP which merges codes of two objects (or object groups) $x_1, x_2$ and the relative position between their OBBs $r_{x_1x_2}$ into one single code $y$. In our hierarchy, we stipulate that the first child $x_1$ supports the second child $x_2$. The encoder is formulated as:

$$y = f_{W_{\text{Se}}, b_{\text{Se}}}(\lfloor x_1, x_2, r_{x_1x_2} \rfloor)$$

The corresponding decoder $\text{SuppDec}$ splits the parent code $y$ back to its children $x'_1$ and $x'_2$ and the relative position between them $r'_{x'_1x'_2}$, using a reverse mapping as shown below:

$$\lfloor x'_1, x'_2, r'_{x'_1x'_2} \rfloor = f_{W_{\text{Sd}}, b_{\text{Sd}}}(y)$$

**Surround** $\text{SurrEnc}$, the encoder for the surround relation is an MLP which merges codes for three objects $x_1, x_2, x_3$ and two relative positions $r_{x_1x_2}, r_{x_1x_3}$ into one single code $y$. In surround relation, we have one central object around which the other two objects are placed on either side. So we calculate the relative position for the two surrounding objects w.r.t the central object. This module can be extended to take more than three children nodes but in our case, we only consider three. The encoder is formulated as:

$$y = f_{W_{\text{SRe}}, b_{\text{SRe}}}(\lfloor x_1, x_2, x_3, r_{x_1x_2}, r_{x_1x_3} \rfloor)$$

The corresponding decoder $\text{SurrDec}$ splits the parent code $y$ back to children codes $x'_1, x'_2,$ and $x'_3$ and their relative positions $r'_{x'_1x'_2}, r'_{x'_1x'_3}$ using a reverse mapping, as shown below:

$$\lfloor x'_1, x'_2, x'_3, r'_{x'_1x'_2}, r'_{x'_1x'_3} \rfloor = f_{W_{\text{SRd}}, b_{\text{SRd}}}(y)$$

**Co-occurrence** The encoder for the co-occurrence relation, $\text{Co-ocEnc}$, is an MLP which merges codes for two objects (or object groups) $x_1, x_2$ and the relative position $r_{x_1x_2}$ between them into one single code $y$. In our structural scene hierarchies, the first child $x_1$ corresponds to the object (or object group) with the larger OBB, than that of $x_2$. The $\text{Co-ocEnc}$ is formulated as:

$$y = f_{W_{\text{COe}}, b_{\text{COe}}}(\lfloor x_1, x_2, r_{x_1x_2} \rfloor)$$

The corresponding decoder $\text{Co-ocDec}$ splits the parent code $y$ back to its children $x'_1$ and $x'_2$ and the relative position between them $r'_{x'_1x'_2}$, using a reverse mapping, given as:

$$\lfloor x'_1, x'_2, r'_{x'_1x'_2} \rfloor = f_{W_{\text{COd}}, b_{\text{COd}}}(y)$$

**Wall** The wall encoder, $\text{WallEnc}$, is an MLP that merges two codes, $x_2$ for an object (or object group) and $x_1$ for the object/group’s nearest wall, along with the relative position $r_{x_1x_2}$, into one single code. In our hierarchy, a wall code is always the left child for the wall
encoder, which is formulated as:

\[ y = f_{W_e, b_e}([x_1 \ x_2 \ r_{x_1 x_2}]) \]

The corresponding decoder \( WallDec \) splits the parent code \( y \) back to its children \( x'_1 \) and \( x'_2 \) and the relative position between them \( r'_{x'_1 x'_2} \) using a reverse mapping, as shown below:

\[ [x'_1 \ x'_2 \ r'_{x'_1 x'_2}] = f_{W_d, b_d}(y) \]

**Root** The final encoder is the root module which outputs the root code. The root encoder, \( RootEnc \), is an MLP which merges codes for 5 objects \( x_1, x_2, x_3, x_4, x_5 \) and 4 relative positions \( r_{x_1 x_2}, r_{x_1 x_3}, r_{x_1 x_4} \) and \( r_{x_1 x_5} \) into one single code. In our scene hierarchies, floor is always the first child \( x_1 \) and the remaining four child nodes correspond to four walls nodes in an anti-clockwise ordering (as seen from the top-view). The root encoder is formulated as:

\[ y = f_{W_e, b_e}([x_1 \ x_2 \ r_{x_1 x_2} \ x_3 \ r_{x_1 x_3} \ x_4 \ r_{x_1 x_4} \ x_5 \ r_{x_1 x_5}]) \]

The corresponding decoder \( RootDec \) splits the parent code \( y \) back to child codes \( x'_1, x'_2, x'_3, x'_4 \) and \( x'_5 \) and the 4 relative positions \( r'_{x'_1 x'_2}, r'_{x'_1 x'_3}, r'_{x'_1 x'_4}, r'_{x'_1 x'_5} \) using a reverse mapping, as shown below:

\[ [x'_1 \ x'_2 \ r'_{x'_1 x'_2} \ x'_3 \ r'_{x'_1 x'_3} \ x'_4 \ r'_{x'_1 x'_4} \ x'_5 \ r'_{x'_1 x'_5}] = f_{W_d, b_d}(y) \]

The dimensions of the hidden and output layers for \( RootEnc \) and \( RootDec \) are 1,050 and 350, respectively. For the other modules, the dimensions are 750 and 250, respectively. This is because the root node has to accommodate more information compared to other nodes.

Lastly, we jointly train an auxiliary node classifier, \( NodeClsfr \), to determine which module to apply at each recursive decoding step. This classifier is a neural network with one hidden layer that takes as input the code of a node in the hierarchy, and outputs whether the node represents a box, support, co-occurrence, surround or wall node. Depending on the output of the classifier, either \( WallDec, SuppDec, SurrDec, Co-ocDec \) or \( BoxDec \) is invoked.

**Training** To train our RvNN-VAE network, we randomly initialize the weights sampled from a Gaussian distribution. Given a training hierarchy, we first encode each leaf-level object using \( BoxEnc \). Next, we recursively apply the corresponding encoder (\( SuppEnc, SurrEnc, Co-ocEnc, WallEnc \) or \( RootEnc \)) at each internal node until we obtain the root. The root codes are approximated to a Gaussian distribution by the VAE. A code is randomly sampled from this distribution and fed to the decoder. Finally, we invert the process, by first applying the \( RootDec \) to the sampled root vector, and then recursively applying the decoders \( WallDec, SuppDec, SurrDec \) and \( Co-ocDec \), followed by a final application of \( BoxDec \) to recover the leaf vectors.
The reconstruction loss is formulated as the sum of squared differences between the input and output leaf vectors and the relative position vectors. The total loss is then formulated as the sum of reconstruction loss and the classifier loss, and Kullback-Leibler (KL) divergence loss for VAE. We simultaneously train NodeClsfr, with a five class softmax classification with cross entropy loss to infer the tree topology at the decoder end during testing.

Testing During testing, we must address the issue of decoding a given root code to recover the constituent bounding boxes, relative positions, and object labels. To do so, we need to infer a plausible decoding hierarchy for a new scene. To decode a root code, we recursively invoke the NodeClsfr to decide which decoder to be used to expand the node. The corresponding decoder (WallDec, SuppDec, SurrDec, Co-ocDec or BoxDec) is used to recover the codes for the children nodes until the full hierarchy has been expanded to the leaves. Since the leaf vectors output by BoxDec are continuous, we convert them to one-hot vectors by setting the maximum value to one and the rest to zeros. In this way, we decode the root code into a full hierarchy and generate a scene based on the inferred hierarchy.

Constructing the final scene After decoding, we need to transform the hierarchical representation to 3D indoor scenes. With the object information present in leaf nodes and relative position information at every internal node, we first recover the bounding boxes for the non-leaf (internal) nodes in a bottom-up manner. By setting a specific position and orientation for the floor, we can compute the placement of the walls and their associated object groups, aided by the relative position information available at every internal node. When we reach the leaf level with a top-down traversal, we obtain the placement of each object, as shown in Figure 5.2(right most). The labeled OBBs are then replaced with 3D objects retrieved from a 3D shape database based on object category and model dimensions.

3.6 Results, evaluation, and applications

In this section, we explain our experimental settings, show results of scene generation, evaluate our method over different options of network design and scene representation, and make comparisons to close alternatives. The evaluation has been carried out with both qualitative and quantitative analyses, as well as perceptual studies. Finally, we demonstrate several applications.

Dataset and training. The training data for our method is extracted from SUNCG [127], a large and diverse collection of realistic indoor scenes encompassing scene types such as bedroom, living room, kitchen, and office. In our experiments, we only consider rooms with a floor and rectangular wall boundaries, without considering wall-mounted objects. Also, to facilitate the network training, we remove the most infrequently appearing object categories over all scenes in every room type, as well as rooms with too few or too many
<table>
<thead>
<tr>
<th>Room type</th>
<th># Scenes</th>
<th># Object categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bedroom</td>
<td>18,763</td>
<td>20</td>
</tr>
<tr>
<td>Living room</td>
<td>4,440</td>
<td>24</td>
</tr>
<tr>
<td>Kitchen</td>
<td>5,974</td>
<td>14</td>
</tr>
<tr>
<td>Office</td>
<td>3,774</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 3.1: Some statistics for the training dataset used in our experiments. The object categories included in the table cover the most frequently appearing objects in SUNCG. The remaining, infrequent, objects were removed.

![Scene structure diagram](image)

Figure 3.9: A scene generated by our method and its corresponding decoded hierarchy, which defines the scene structure.

Using our method, a 3D indoor scene can be generated from a random vector by our trained RvNN-VAE network. The vector is first decoded into a structural scene hierarchy (e.g., see Figure 3.9), and then converted into a scene via object retrieval. We shall demonstrate that the scenes generated by our method are plausible and diverse, while exhibiting some levels of novelty.

3.6.1 Scene generation results

Using our method, a 3D indoor scene can be generated from a random vector by our trained RvNN-VAE network. The vector is first decoded into a structural scene hierarchy (e.g., see Figure 3.9), and then converted into a scene via object retrieval. We shall demonstrate that the scenes generated by our method are plausible and diverse, while exhibiting some levels of novelty.

To assess the diversity and novelty of the scenes, we need a scene similarity measure. To this end, we adopt graph kernels [?], whose definition depends on object labels, sizes, and object-object relations in a scene. Figure 3.10 shows the three nearest neighbors, via graph
Figure 3.10: Top three closest scenes from the training set, based on graph kernel, showing diversity of scenes from the training set.

kernels, in the training dataset to a given scene. These results are quite representative and indicative of the diversity of the training scenes, which makes the learning problem more challenging.

**Scene generation** Figure 5.10 presents a sampling of bedrooms generated by our method, which were randomly selected from a pool of 1,000 generated scenes, over three levels of scene complexity based simply on object counts. We can first observe that these scenes are plausible as exemplified by proper object placements against the walls and frequent object co-occurrences, e.g., TV+TV stand, bed+nightstands, desk+chair+computer combinations, etc. To demonstrate that our generative network does not simply memorize training scenes, we show the closest scene, based on graph kernels, from the training set. Also, we show the closest scene from the pool of generated scenes to assess diversity of the scenes generated. Figure 3.12 shows several scenes generated for the other room types. More results can be found in the Appendix (Section 3.8).

**Timing** After training, our method can generate a 3D scene in a fraction of a second. In comparison, recent work by Wang et al. [139] reports a generation time of 4 minutes. More specifically, our generative network was able to produce 10K bedroom scene hierarchies in a total of 94 seconds on a GPU. Then, it took 933 seconds to convert the hierarchies to 3D scenes with object placements in MATLAB running on a CPU. Hence, it takes 0.1027s to generate a 3D bedroom scene, on average. The timings are reported on a machine with one NVIDIA GeForce GTX 1080 Ti GPU and an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, with 64GB of memory. The training time depends on the size of the dataset and scene
Figure 3.11: Bedrooms generated by our method (a), in comparison to (b) closest scene from the training set, to show novelty, and to (c) closest scene from among 1,000 generated results, to show diversity. Different rows show generated scenes at varying complexity, i.e., object counts.

Complexity. Running 500 epochs, it took 36 hours to train the RvNN-VAE for bedrooms and about 7-12 hours for the other room types.

**Object co-occurrence** One possible way to validate our learning framework is to examine its ability to reproduce the probabilities of object co-occurrences in the training scenes. For this purpose, we define a conditional probability for two object categories $c_1$ and $c_2$: $P(c_1|c_2) = \frac{N(c_1,c_2)}{N(c_2)}$, where $N(c_1,c_2)$ is the number of scenes with *at least* one object belonging to category $c_1$ and one object belonging to category $c_2$, and $N(c)$ is the number of scenes with *at least* one object belonging to category $c$. Figure 3.13 plots similarities between the
Figure 3.12: Several living room, kitchen, and office scenes generated by our method. More results can be found in the Appendix (Section 3.8).

$P(c_1|c_2)$’s over object category pairs for bedrooms and living rooms, where the similarity is measured between the conditional probabilities over the entire training set of scenes and 1,000 randomly generated scenes of the same room type. With the exception of the TV-floor lamp pair for living rooms, the strong similarities indicate that our learned RvNN is able to generate scenes with similar object co-occurrences compared to the training scenes.
3.6.2 Comparisons

The most popular approach to 3D scene synthesis has so far been based on probabilistic graphical models \cite{65,113}, which learn a joint conditional distribution of object co-occurrence from an exemplar dataset and generate novel scenes by sampling from the learned model.

To compare with the state-of-the-art, we take the closest work of Kermani et al.\cite{65} which does not rely on human-activities or other pre-defined priors. In order to make a fair comparison, we implement their algorithm on the same set of scenes from SUNCG dataset that were used to train our RvNN-VAE network. In Figure 3.14, we present three randomly selected synthesized scenes using their approach and ours, with the same set of object models. Their method generates a valid set of objects but produces many invalid arrangements and object orientations, affecting the plausibility of synthesized scenes. These are a result of their relative position encoding scheme and the underlying Monte-Carlo Markov Chain (MCMC) sampling model, which requires thousands of iterations to generate one plausible scene. Another drawback of using sampling based techniques for scene synthesis is that there is no guarantee on exact alignment between objects in the synthesized scenes, unlike ours. While our method generates large number of plausible, diverse scenes in seconds (e.g., 10K bedroom scenes in 94 seconds), it takes anywhere from a few minutes to tens of minutes to synthesize one scene (depending upon the complexity) using their approach.

To better compare the two methods, we perform a subjective analysis over the scenes generated by both methods, as well as those from the training set. In particular, we design two perceptual studies. In the first study, we ask the participants to select from a triplet of scenes the most plausible one. The triplet consists of scenes from the training set and
results generated by the two methods (ours and [65]), presented in a random order. In the second study, the same set of participants are asked to rate each of the three scenes in the triplets based on their plausibility, on a scale of 1 to 5 (5 being most plausible). The two studies are conducted separately, to minimize their mutual influence. The study employed 50 participants with different backgrounds - 30 graphics/vision researchers, 10 non-graphics researchers and 10 non-technical subjects. Each participant is presented with 10 sets of triplets. Therefore, we collect feedbacks from 50 participants on $10 \times 3$ scenes for each study.

Figure 3.15 and 3.16 plot the statistics from the perceptual studies. From Figure 3.15, we see that the participants mostly think that the training scenes are more plausible, since these scenes are human designed. The scenes generated by our method are also frequently chosen, especially by Graphics/Vision researchers, achieving a comparable level with training scenes. Figure 3.16 shows the statistics of plausibility rating. We observe a comparable rating for scenes from the training set and our results, demonstrating the high quality of our results.

We also ask for free-text responses from the participants to understand their basis of making a choice in the studies. From the feedback, participants without technical background tend to judge the scene plausibility based on their intuition on real world scenes. Therefore, they are more sensitive to object shapes (e.g., relative size) than object relations and overall layout.
Figure 3.15: Statistics on plausible scene selection from the perceptual studies: The participants are asked to select the most plausible scene for each triplet, consisting of bedroom scenes from the training set, our results and the results from [65]. The plots show the percentage of rated plausible scenes for each method.

On the other hand, Graphics/Vision researchers pay more attention to object placements such as their orientation and their relative positions. This is why they tend to give slightly higher scores to all scenes. Moreover, non-(Graphics/Vision) researchers tend to give higher scores to our results even though they believed the scenes from the training set are generally more plausible than ours in the first study.

Comparison against Concurrent Works. Recent works of Qi et al. [113] and Wang et al. [139] use SUNCG dataset to solve a similar problem but are tuned for different set of inputs. The work of [113] uses object sizes, object positions, orientations and human positions interacting with entities in a scene as inputs to their method and run MCMC sampling to synthesize new scene layouts, whereas [139] input a partial scene geometry to synthesize new layouts learned from convolution neural networks trained on 2D top-down images of 3D scenes. We further evaluate by performing pairwise perceptual studies on the plausibility of generated/synthesized scenes against these two works.

In Figure 3.17, we show the results of plausibility studies of our generated scenes against those synthesized from Qi et al. [113]. We do a forced choice, pairwise comparison on 15 pairs of scenes, with 5 scene pairs each from 3 scene types, viz., bedrooms, living rooms and offices. We take the synthesized scenes from Qi et al. [113] and retrieve scenes from
Figure 3.16: Results of plausibility rating by participants for bedrooms: The participants are asked to rate a scene based on their plausibility on a scale of 1 to 5 (5 being the most plausible). The values are the average scores for scenes from the training set, our results and the results by [65].

our generated set based on the category count of the objects. For consistency, we do not take into account doors and windows. We adjust the scenes to have same/similar object models. All the pairs are rendered from the same viewpoint and presented in a random order to 20 participants involved in the perceptual studies. From the figure, we see that our generated scenes receive 63% of the total votes since object arrangements in our generated scenes are better than Qi et al. [113]. This suggests that there’s a higher uncertainty in the convergence of the scene layout using their method, as it relies on MCMC sampling. In addition, it takes around 22 minutes to synthesize one scene from their method as opposed to 102.7 ms using ours.

We again refer to Figure 3.17 to show the results of studies against [139]. With the same settings as before, except that the synthesized scenes now come from [139], we collect responses from the participants for this study. We observe from the figure that our generated scenes are slightly preferred over scenes synthesized from [139], receiving 56% of the total votes. The two methods are more or less comparable in terms of plausibility of the synthesized scenes but their method cannot produce object-on-furniture supported scenes. As a result, their scenes tend to be less preferred than ours. In addition, it takes around 4 minutes to synthesize one scene using their method.
Figure 3.17: Statistics on plausible scene selection from the pairwise perceptual studies: The participants are asked to select the most plausible scene in a pair, consisting of scenes from bedrooms, living and offices, presented in a random order. The plots show the percentage of rated plausible scenes for synthesized scenes from the two methods against ours.

3.6.3 Validation of network design and scene encoding

To validate several important choices in our network design and scene encoding, we show experimental results with vs. without these design decisions. Note that these choices also highlight key differences between GRAINS and GRASS [79], which is the prior RvNN-VAE designed for learning shape structures.

**Wall and root encoders/decoders**  By designating wall and root encoders and decoders with the last grouping operations in our RvNN and carrying them out in a fixed order, we follow the premise that walls should serve a similar role as floors, i.e., to “ground” the placements of objects in a room. Treating walls and floors just like any other objects, i.e., without wall or root encoders and decoders, would lead to an RvNN that is much like GRASS but with symmetry and assembly operations between shape parts replaced by object support, co-occurrences, etc. Under this setting, walls are merely objects that have a high occurrence frequency. It is difficult to train the network to always produce the correct number and placements of walls, as shown by a typical result in Figure 3.18(a). In fact, only 36.9% of 1K generated scenes have exactly four walls, with most of these rooms having incorrect wall placements in the layout.
Figure 3.18: Typical bedroom scenes generated by our RvNN-VAE with various options involving wall and root encoders and decoders. (a) Without wall or root encoders and decoders. (b) With wall encoders and decoders, but not for root. (c) With both types of encoders and decoders.

In Figure 3.18(b), we show a typical scene generated using our method where wall encoders and decoders are incorporated, but not the root encoder. As we can observe, this time the network is able to properly learn the positioning of objects against each wall. However, without the root encoder, which accounts for and learns the relative positions of the four walls with respect to the floor, the placements of the walls can be quite off. Without the designated root encoder, we must resort to some alternative way to merge the walls and floors. When generating the result shown in Figure 3.18(b), we applied the wall encoders to merge two opposite walls, and then the two pairs of walls. Finally, the combined node representing four walls is merged with the floor using a support relation.

**Encoding of relative object positions** With the key observation that predictabilities in object positions in a 3D scene are manifested in relative, and not absolute, terms, we encode relative object positions using angles and offsets, which are complemented with attachment and alignment bits to reinforce placement precisions. To contrast this with the use of absolute object positions in the network, we show in the first row of Figure 3.19 several typical bedrooms generated with this latter option. We can see that our RvNN-VAE was unable to learn plausible object placements: most objects are placed somewhat randomly without proper attachments or alignments, due to the lack of predictability in their absolute positions.

In the second row of Figure 3.19, we show results of scene generation using our relative position encoding but without alignment or attachment bits. As expected, while the rough relative positioning of objects may be plausible, e.g., nightstands do surround the bed, precise alignments and attachments, e.g., bed against the wall or nightstands against the bed, are not fulfilled.

In the final row of Figure 3.19, we show scenes generated using our RvNN-VAE, with relative position encoding as described in Section 3.4.2, except that the position of the
target box is encoded using a *single* translation vector between the centers of the target and reference boxes. The results show that this alternative is problematic since box-to-box translations can vary greatly, e.g., dressers can be placed in all directions around a bed. In such cases, the trained network has the tendency to generate “average” translation vectors. This is why the dressers tend to be placed near the center of the beds. In contrast, our representation which utilizes a binary indicator vector to identify the closest edge pairs for offset encoding avoids the generation of average offset values.
Figure 3.20: Typical bedroom scenes generated by our method when semantic labels are not included in object encodings in the leaf vectors, where 3D objects were retrieved based only on geometric information about the OBBs.

**Semantic labels** We incorporate semantic labels into the object encodings since object co-occurrences are necessarily characterized by semantics, not purely geometric information such as OBB dimensions and positions. As expected, removing semantic labels from the object encodings would make it difficult for the network to learn proper object co-occurrences. As well, without semantic information associated with the OBB’s generated by the RvNN-VAE, object retrieval based on OBB geometries would also lead to clearly implausible scenes, as shown by examples in Figure 3.20.

### 3.6.4 Applications

We present three representative applications of our generative model for 3D indoor scenes, highlighting its several advantages: 1) cross-modality scene generation; 2) generation of scene hierarchies; and 3) fast synthesis of large volumes of scene data.

**2D layout guided 3D scene modeling.** Several applications such as interior design heavily use 2D house plan, which is a top-view 2D box layout of an indoor scene. Automatically generating a 3D scene model from such 2D layout would be very useful in practice. Creating a 3D scene from labeled boxes would be trivial. Our goal is to generate a series of 3D scenes whose layout is close to the input boxes without semantic labels, while ensuring a plausible composition and placement of objects. To do so, we encode each input 2D box into a leaf vector with unknown class (uniform probability for all object classes). We then construct a hierarchy based on the leaves, obtaining a root code encoding the entire 2D layout. Note that the support relation changes to overlap in 2D layout encoding. The root code is then mapped into a Gaussian based on the learned VAE. A sample from the Gaussian can be decoded into a hierarchy of 3D boxes with labels by the decoder, resulting in a 3D scene. Thus, a set of scenes can be generated, whose spatial layouts closely resemble the input 2D layout. A noteworthy fact is that both the encoder and decoder used here are pretrained on 3D scenes; no extra training is required for this 2D-to-3D generation task. This is because
Figure 3.21: 3D scene generation from 2D box layouts. Our trained RvNN-VAE can convert a 2D box layout representing the top-view of a scene (a) into a root code and then decode it into a 3D scene structure that closely resembles the 2D layout. (b) shows scenes decoded from the mean of the Gaussian of the latent vector and (c) shows scenes generated from latent vectors randomly sampled from the Gaussian.

Figure 3.22: Hierarchy-guided scene editing. Given a generated scene and its hierarchy (a), user can select a subtree (colored in green or blue) corresponding to a sub-scene (objects colored similarly). The subtree (sub-scene) can be replaced by another from other generated hierarchies, resulting in an edited scene (b).

2D box layout is equivalent to 3D one in terms of representing spatial layout and support relations, which makes the learned network reusable for cross-modality generation.

Figure 3.21 shows a few examples of 2D layout guided 3D scene generation, where no label information was used. The missing label information is recovered in the generated scenes. In the generated 3D scenes, many common sub-scenes observed from the training dataset are preserved while fitting to the input 2D layout.

**Hierarchy-guided scene editing** During scene modeling, it is desirable for the users to edit the generated scenes to reflect their intent. With the hierarchies accompanied by the generated scenes, our method enables the user to easily edit a scene at different granularities...
Table 3.2: Performance of semantic scene segmentation using PointNet trained on different datasets. Our generated scenes can be used to augment SUNCG to boost performance.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Ours, 24K</th>
<th>SUNCG, 12K + Ours, 24K</th>
<th>SUNCG 24K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>63.45%</td>
<td>70.08%</td>
<td>77.04%</td>
</tr>
</tbody>
</table>

Table 3.2 shows the average accuracy on scene segmentation task. We see that training using our generated scenes produces comparable performance to that using scenes from SUNCG which are manually created. Using only our generated scenes as the training data gains higher accuracy than that using the same amount of SUNCG data. This is probably because we have more repeated objects and object groups in our generated scenes and the diversity of object shapes in our generated scenes is not as large as in SUNCG scenes. More importantly, enhancing SUNCG data with scenes generated by our network helps the learned model generalize better and boost its performance.

### 3.7 Discussion, Limitations, and Future Work

Our work makes a first attempt at developing a generative neural network to learn hierarchical structures of 3D indoor scenes. The network, coined GRAINS, integrates a recursive
neural network with a variational autoencoder, enabling us to generate a novel, plausible 3D scene from a random vector in less than a second. The network design consists of several unique elements catered to learning scene structures for rectangular rooms, e.g., relative object positioning, encoding of object semantics, and use of wall objects as initial references, distinguishing itself from previous generative networks for 3D shapes [150, 79].

As shown in Section 3.6.3, the aforementioned design choices in GRAINS are responsible for improving the plausibility of the generated scenes. However, one could also argue that they are introducing certain “handcrafting” into the scene encoding and learning framework. It would have been ideal if the RvNN could: a) learn the various types of object-object relations on its own from the data instead of being designed with three pre-defined grouping operations; b) rely only on the extracted relations to infer all the object semantics, instead of encoding them explicitly; and c) work with a more generic representation of object positions than our current special-purpose encoding of relative positionings. At this point, these are all tall challenges to overcome. An end-to-end, fully convolutional network that can work at the voxel level while being capable of generating clean and plausible scene structures also remains elusive.

Another major limitation of GRAINS, and of generative recursive autoencoders such as GRASS [79], is that we do not have direct control over objects in the generated scene. For example, we cannot specify object counts or constrain the scene to contain a subset of objects. Also, since our generative network produces only labeled OBBs and we rely on a fairly rudimentary scheme to retrieve 3D objects to fill the scene, there is no precise control over the shapes of the scene objects or any fine-grained scene semantics, e.g., style compatibility. It is possible to employ a suggestive interface to allow the user to select the final 3D objects. Constrained scene generation, e.g., by taking as input a partial scene or hierarchy, is also an interesting future problem to investigate.

Modeling indoor scene structures in a hierarchical way does hold its merits, as demonstrated in this work; it is also a natural fit to RvNNs. However, hierarchies can also be limiting, compared to a more flexible graph representation of scene object relations. For example, it is unclear whether there is always a sensible hierarchical organization among objects which may populate a messy office desk including computer equipments, piles of books, and other office essentials. Also, in many instances, it can be debatable what the best hierarchy for a given scene is. To this end, recent works on learning generative models of graphs may be worth looking into.

Finally, as a data-driven method, GRAINS can certainly produce unnatural object placements and orientations. A few notable failure cases can be observed in Figure 5.10, e.g., a computer display and a swivel chair facing the wall, shown in the fifth row of column (a). One obvious reason is that the training data is far from perfect. For example, object placements in some of the SUNCG scenes can be unnatural, as shown in the last row of column (b), where a bed is placed in the middle of the room. More critically, the training
scene hierarchies in our work were produced heuristically based on spatial object relations and as such, there is no assurance of consistency even among similar scenes belonging to the same scene category. In general, computing consistent hierarchies over a diverse set of scenes is far from straightforward. Comparatively, annotations for pairwise or group-wise object relations are more reliable. An intriguing problem for future work is to develop a generative network, possibly still based on RvNNs and autoencoders, that are built on learning subscene structures rather than global scene hierarchies.

3.8 Appendix

More scene generation results We present more scene generation results for living rooms, kitchens, and offices in figure 3.26. We train a separate RvNN-VAE for each room category, using the same settings as mentioned in the main paper. Our network learns the common patterns present in various sub-scenes, in each room category; for example: (a) sofa and table in living rooms, (b) desk, computer and chair in offices, and (c) set of attached kitchen cabinets in kitchens. Note that for kitchens, the training set is biased towards kitchen-cabinets, which limits the variety of object classes in the generated scenes. The key characteristic, however, is the attachment of kitchen-cabinets, which are reflected ably by our generated scenes. Our RvNN-VAE framework not only learns to preserve the object patterns in the sub-scenes (thus making the generated scenes plausible), but also enables diversity by incorporating multiple sub-scenes into a room and placing them accordingly.

Object co-occurrence One possible way to validate our learning framework is to examine its ability to reproduce the probabilities of object co-occurrence in the training scenes. To measure the co-occurrence, we define a conditional probability $P(c_1|c_2) = \frac{N(c_1,c_2)}{N(c_2)}$, where $N(c_1,c_2)$ is the number of scenes with at least one object belonging to category $c_1$ and one object belonging to category $c_2$. Figure 3.27 plots the object co-occurrences in the training scenes (top row) and our generated scenes (bottom row). Figure 3.28 is a similarity plots between the training set and the RvNN-VAE generated results purples$(c_1|c_2) = 1 - |P_t(c_1|c_2) - P_g(c_1|c_2)|$. After training, our network is able to generate scenes with similar object occurrences compared to the training scenes.

Distribution of the relative positions To further support our learning framework, we observe the distribution of relative positions between relevant objects. Conforming to common positioning patterns would reflect a good framework. Figure 3.29 shows the distributions of relative positions between object categories, from the training set (first column) and scenes generated using different relative position formats(ours, absolute position and box transforms). The first row shows the distribution of the relative positions of nightstands against the nearest bed. The blue dots represent the center points of the nightstands, and
Figure 3.23: (a) Generated living rooms

Figure 3.24: (b) Generated offices

Figure 3.25: (c) Generated kitchens

Figure 3.26: More results of other categories. Top two rows: generated living rooms; middle two rows: generated offices; last two rows: generated kitchens.
Figure 3.27: The object co-occurrence in the training scenes (top row) and our generated scenes (bottom row). From left to right are the plots of bedroom, living room, office, kitchen. Higher probability values correspond to warmer colors. Note that we only plot the co-occurrence between objects belonging to the frequent classes.

Figure 3.28: Similarity plots between object co-occurrence in the training set vs. in the RvNN-generated scenes. purpleLeft to right: bedroom, living room, office and kitchen.
Figure 3.29: The relative position distribution. The relative positions are obtained from: (a) indoor scenes from the training set; generated scenes each using (b) our relative position format, (c) absolute position format and (d) box transforms, respectively. The first row shows the center points of stands (blue dots) and beds (red rectangles), under the local frames of the beds. The second row shows the center points of chairs (blue dots) and desks (red rectangles), under the local frames of the desks. The arrows show the front orientations of beds and desks. Best viewed in color on Adobe Reader.

diagram...

the red boxes represent the beds. The orientations of the beds are given by the arrow on the top right corner. For the bed-nightstand pair, our method outperforms the other two relative positions as it learns to put most of the nightstands on the left and/or right side of the beds. Consider the chair-desk pair shown in the second row of figure 3.29, the training set has most chairs in front of the desks and also has some chairs in the back of the desks. But our generated scenes rarely place a chair behind a desk. In addition, we find that some of the scenes have chairs on the sides of the desks. This knowledge is inferred from the placement of object-object pairs across all scenes in a room category.
4.1 Introduction

Two-dimensional layouts are ubiquitous visual abstractions in graphic and architectural designs. They typically represent blueprints or conceptual sketches for such data as floorplans, documents, scene arrangements, and UI designs. Recent advances in pattern analysis and synthesis have propelled the development of generative models for layouts [35, 78, 181, 47, 80] and led to a steady accumulation of relevant datasets [182, 151, 31, 180]. Despite these developments however, there have been few attempts at employing a deeply learned metric to reason about layout data, e.g., for retrieval, data embedding, and evaluation. For example, current evaluation protocols for layout generation still rely heavily on segmentation metrics such as intersection-over-union (IoU) [47, 93] and human judgement [47, 80].

The ability to compare data effectively and efficiently is arguably the most foundational task in data analysis. The key challenge in comparing layouts is that it is not purely a task of visual comparison — it depends critically on inference and reasoning about structures, which are expressed by the semantics and organizational arrangements of the elements or subdivisions which compose a layout. Hence, none of the well-established image-space metrics, whether model-driven, perceptual, or deeply learned, are best suited to measure structural layout similarity. Frequently applied similarity measures for image segmentation such as IoUs and F1 scores all perform pixel-level matching “in place” — they are not structural and can be sensitive to element misalignments which are structure-preserving.

In this work, we develop a deep neural network to predict structural similarity between two 2D layouts, e.g., floorplans or UI designs. We take a predominantly structural view of layouts for both data representation and layout comparison. Specifically, we represent each layout using a directed, fully connected graph over its semantic elements. Our network learns structural layout similarity via neural graph matching, where an attention-based graph matching network [82] is designed under a triplet network setting. The network, coined
Figure 4.1: LayoutGMN learns a structural layout similarity metric between floorplans and other 2D layouts, through attention-based neural graph matching. The learned attention weights (numbers shown in the boxes) can be used to match the structural elements.

LayoutGMN, takes as input a triplet of layout graphs, composed together by one pair of anchor-positive and one pair of anchor-negative graphs, and performs intra-graph message passing and cross-graph information communication per pair, to learn a graph embedding for layout similarity prediction. In addition to returning a metric, the attention weights learned by our network can also be used to match the layout elements; see Figure 5.1.

To train our triplet network, it is natural to consider human labeling of positive and negative samples. However, it is well-known that subjective judgements by humans over structured data such as layouts are often unreliable, especially with non-experts [178, 6]. When domain experts are employed, the task becomes time-consuming and expensive [178, 6, 46, 30, 67, 149], where discrepancies among even these experts still remain [46]. In our work, we avoid this issue by resorting to weakly supervised training of LayoutGMN, which obtains positive and negative labels from the training data through thresholding using layout IoUs [93].

The motivations behind our network training using IoUs are three-fold, despite the IoU’s shortcomings for structural matching. First, as one of the most widely-used layout similarity measures [93, 47], IoU does have its merits. Second, IoUs are objective and much easier to obtain than expert annotations. Finally and most importantly, our network has a built-in inductive bias to enforce structural correspondence, via inter-graph information exchange, when learning the graph embeddings. The inductive bias results from an attention-based graph matching mechanism, which learns structural matching between two graphs at the node level (Eq 4.3, 4.6). Such a structural bias can effectively compensate for the lack of structure awareness in the IoU-based triplet loss during training. In Figure 4.2, we illustrate the effect of this structural bias on the metric learned by our network. Observe that the last two layouts are more similar structurally than the first two. This is agreed with by our metric LayoutGMN, but not by IoU feedback.

We evaluate our network on retrieval tasks over large datasets of floorplans and UI designs, via Precision@$k$ scores, and investigate the stability of the proposed metric by
Figure 4.2: Structure matching in LayoutGMN “neutralizes” IoU feedback. In each example (left: floorplan; right: UI design), a training sample $N$ labeled as “Negative” by IoU is more structurally similar to the anchor ($A$) than $P$, a “Positive” sample. With structure matching, our network predicts a smaller $A$-to-$N$ distance than $A$-to-$P$ distance in each case, which contradicts IoU.

Checking retrieval consistency between a query and its top-1 result, over many such pairs; see Sec. 4.5.2. Overall, retrieval results by LayoutGMN better match human judgement of structural layout similarity compared to both IoUs and other baselines including a state-of-the-art method based on graph neural networks [93]. Finally, we show a label transfer application for floorplans enabled by the structure matching learned by our network (Sec 4.5.5).

4.2 Related Work

**Layout analysis.** Early works [63, 7] on document analysis involved primitive heuristics to analyse document structures. Organizing a large collection of such structures into meaningful clusters requires a distance measure between layouts, which typically involved content-based heuristics [116] for documents and constrained graph matching algorithm for floorplans [148]. An improved distance measure relied on rich layout representation obtained using autoencoders [20, 90], operating on an entire UI layout. Although such models capture rich raster properties of layout images, layout structures are not modeled, leading to noisy recommendations in contextual search over layout datasets.

**Layout generation.** Early works on synthesizing 2D layouts relied on exemplars [54, 74, 128] and rule-based heuristics [104, 129], and were unable to capture complex element distributions. The advent of deep learning led to generative models of layouts of floorplans [151, 47, 16, 102], documents [78, 35, 181], and UIs [20, 19]. Perceptual studies aside, evaluation of generated layouts, in terms of diversity and generalization, has mostly revolved around IoUs of the constituent semantic entities [78, 35, 47]. While IoU provides a visual similarity measure, it is expensive to compute over a large number of semantic entities, and is sensitive to element positions within a layout. Developing a tool for structural comparison
would perhaps complement visual features in contextual similarity search. In particular, a learning-based method that compares layouts structurally can prove useful in tasks such as layout correspondence, component labeling and layout retargeting. We present a Layout Graph Matching Network, called LayoutGMN, for learning to compare two graphical layouts in a structured manner.

**Structural similarity in 3D.** Fisher et al. [28] develop Graph Kernels for characterizing structural relationships in 3D indoor scenes. Indoor scenes are represented as graphs, and the Graph Kernel compares substructures in the graphs to capture similarity between the corresponding scenes. A challenging problem of organizing a heterogeneous collection of such 3D indoor scenes was accomplished in [157] by focusing on a subscene, and using it as a reference point for distance measures between two scenes. Shape Edit Distance, SHED, [73] is another fine-grained sub-structure similarity measure for comparing two 3D shapes. These works provide valuable cues on developing an effective structural metric for layout similarity. Graph Neural Networks (GNN) [83, 72, 8, 120] model node dependencies in a graph via message passing, and are the perfect tool for learning on structured data. GNNs provide coarse-level graph embeddings, which, although useful for many tasks [133, 1, 61, 66], can lose useful structural information in contextual search, if each graph is processed in isolation. We make use of Graph Matching Network [82] to retain structural correspondence between layout elements.

**GNNs for structural layout similarity.** To the best of our knowledge, the recent work by Manandhar et al. [93] is the first to leverage GNNs to learn structural similarity of 2D graphical layouts, focusing on UI layouts with rectangular boundaries. They employ a GCN-CNN architecture on a graph of UI layout images, also under an IoU-trained triplet network [45], but obtain the graph embeddings for the anchor, positive, and negative graphs independently.

In contrast, LayoutGMN learns the graph embeddings in a dependent manner. Through cross-graph information exchange, the embeddings are learned in the context of the anchor-positive (respectively, the anchor-negative) pair. This is a critical distinction to GCN-CNN [93], while both train their triplet networks using IoUs. However, since IoU does not involve structure matching, it is not a reliable measure of structural similarity, leading to labels which are considered “structurally incorrect”; see Figure 4.2.

In addition, our network does not perform any convolutional processing over layout images; it only involves eight MLPs, placing more emphasis on learning finer-scale structural variations for graph embedding, and less on image-space features. We clearly observe that the cross-graph communication module in our GMNs does help in learning finer graph embeddings than the GCN-CNN framework [93]. Finally, another advantage of moving away from any reliance on image alignment is that similarity predictions by our network are more robust against highly varied, non-rectangular layout boundaries, e.g., for floorplans.
4.3 Method

The Graph Matching Network (GMN) [82] consumes a pair of graphs, processes the graph interactions via an attention-based cross-graph communication mechanism and results in graph embeddings for the two input graphs, as shown in Fig 4.4. Our LayoutGMN plugs in the Graph Matching Network into a Triplet backbone architecture for learning a (pseudo) metric-space for similarity on 2D layouts such as floorplans, UIs and documents.

4.3.1 Layout Graphs

Given a layout image of height $H$ and width $W$ with semantic annotations, we abstract each element into a bounding box, which form the nodes of the resulting layout graph. Specifically, for a layout image $I_1$, its layout graph $G_l$ is given by $G_l = (V,E)$, where the node set $V = \{v_1, v_2, \ldots, v_n\}$ represents the semantic elements in the layout, and $E = \{e_{12}, \ldots, e_{ij}, \ldots, e_{n(n-1)}\}$, the edge set, represents the set of edges connecting the constituent elements. Our layout graphs are directed and fully-connected.

**Initial Node Features.** There exist a variety of visual and content-based features that could be incorporated as the initial node features; ex. the text data/font size/font type of an UI element or the image features of a room in a floorplan. For structured learning tasks as ours, we ignore such content-based features and only focus on the box abstractions. Specifically, similar to [35, 39], the initial node features contain *semantic* and *geometric* information of the layout elements. As shown in Fig 4.3, for a layout element $k$ centered at $(x_k, y_k)$, with dimensions $(w_k, h_k)$, its geometric information is:

$$g_k = \begin{bmatrix} x_k \\ y_k \\ w_k \\ h_k \\ w_k h_k \end{bmatrix}.$$
Figure 4.4: LayoutGMN takes two layout graphs as input, performs intra-graph message passing (Eq. 4.2), along with cross-graph information exchange (Eq. 4.3) via an attention mechanism (Eq. 4.5, also visualized in Figure 5.1) to update node features, from which final graph embeddings are obtained (Eq. 4.7).

Instead of one-hot encoding of the semantics, we use a learnable embedding layer to embed a semantic type into a 128-D code, \( s_k \). A two-layer MLP embeds the \( 5 \times 1 \) geometric vector \( g_k \) into a 128-D code, and is concatenated with the 128-D semantic embedding \( s_k \) to form the initial node features \( U = \{u_1, u_2, \ldots, u_n\} \).

**Initial Edge Features.** In visual reasoning and relationship detection tasks, edge features in a graph are designed to capture relative difference of the abstracted semantic entities (represented as nodes) [39, 165]. Thus, for an edge \( e_{ij} \), we capture the spatial relationship (see Fig 4.3) between the semantic entities by a \( 8 \times 1 \) vector:

\[
e_{ij} = \left[ \frac{\Delta x_{ij}}{\sqrt{A_i}}, \frac{\Delta y_{ij}}{\sqrt{A_i}}, \sqrt{\frac{A_j}{A_i}}, U_{ij}, \frac{w_i}{h_i}, \frac{w_j}{h_j}, \sqrt{\Delta x^2 + \Delta y^2}, \frac{\sqrt{W^2 + H^2}}{\sqrt{W^2 + H^2}}, \theta \right],
\]

where \( A_i \) is the area of the element box \( i \); \( U_{ij} = \frac{B_i \cap B_j}{B_i \cup B_j} \) is the IoU of the bounding boxes of the layout elements \( i, j \); \( \theta = \text{atan2}(\frac{\Delta y_{ij}}{\Delta x_{ij}}) \) is the relative angle between the two components, \( \theta \in [-\pi, \pi] \); \( \Delta x_{ij} = x_j - x_i \) and \( \Delta y_{ij} = y_j - y_i \). This edge vector accounts for the translation between the two layout elements, in addition to encoding their box IoUs, individual aspect ratios and relative orientation.

### 4.3.2 Graph Matching Network

The graph matching module employed in LayoutGMN is made up of three parts: (1) node and edge encoders, (2) message propagation layers and (3) an aggregator.
Node and Edge Encoders. We use two MLPs to embed the initial node and edge features and compute their corresponding code vectors:

\[ h_i^{(0)} = MLP_{node}(u_i), \forall i \in U \]
\[ r_{ij} = MLP_{edge}(e_{ij}), \forall (i, j) \in E \] \hspace{1cm} (4.1)

The above MLPs map the initial node and edge features to their 128-D code vectors.

Message Propagation Layers. The graph matching framework hinges on coherent information exchange between graphs to compare two layouts in a structural manner. The propagation layers update the node features by aggregating messages along the edges within a graph, in addition to relying on a graph matching vector that measures how similar a node in one layout graph is to one or more nodes in the other. Specifically, given two node embeddings \( h_i^{(0)} \) and \( h_p^{(0)} \) from two different layout graphs, the node updates for the node \( i \) are given by:

\[ m_{j \rightarrow i} = f_{\text{intra}}(h_i^{(t)}, h_j^{(t)}, r_{ij}), \forall (i, j) \in E_1 \] \hspace{1cm} (4.2)
\[ \mu_{p \rightarrow i} = f_{\text{cross}}(h_i^{(t)}, h_p^{(t)}), \forall i \in V_1, p \in V_2 \] \hspace{1cm} (4.3)
\[ h_i^{(t+1)} = f_{\text{update}} \left( h_i^{(t)}, \sum_j m_{j \rightarrow i}, \sum_p \mu_{p \rightarrow i} \right) \] \hspace{1cm} (4.4)

where \( f_{\text{intra}} \) is an MLP on the initial node embedding code that aggregates information from other nodes within the same graph, \( f_{\text{cross}} \) is a function that communicates cross-graph information, and \( f_{\text{update}} \) is an MLP used to update the node features in the graph, whose input is the concatenation of the current node features, the aggregated information from within, and across the graphs. \( f_{\text{cross}} \) is designed as an Attention-based module:

\[ a_{p \rightarrow i} = \frac{\exp(s_h(h_i^{(t)}, h_p^{(t)}))}{\sum_p \exp(s_h(h_i^{(t)}, h_p^{(t)}))} \] \hspace{1cm} (4.5)
\[ \mu_{p \rightarrow i} = a_{p \rightarrow i} (h_i^{(t)} - h_p^{(t)}) \]

where \( a_{p \rightarrow i} \) is the attention value (scalar) between node \( p \) in the second graph and node \( i \) in the first, and such attention weights are calculated for every pair of nodes across the two graphs; \( s_h \) is implemented as the dot product of the embedded code vectors. The interaction of all the nodes \( p \in V_2 \) with the node \( i \) in \( V_1 \) is then given by:

\[ \sum_p \mu_{p \rightarrow i} = \sum_p a_{p \rightarrow i} (h_i^{(t)} - h_p^{(t)}) = h_i^{(t)} - \sum_p a_{p \rightarrow i} h_p^{(t)} \] \hspace{1cm} (4.6)

Intuitively, \( \sum_p \mu_{p \rightarrow i} \) measures the (dis)similarity between \( h_i^{(t)} \) and its nearest neighbor in the other graph. The pairwise attention computation results in stronger structural bonds
Figure 4.5: Given a triplet of graphs $G_a, G_p$ and $G_n$ corresponding to the anchor, positive and negative examples respectively, the anchor graph paired with each of other two graphs is passed through a Graph Matching Network (Fig 4.4) to get two 1024-D embeddings. Note that the anchor graph has different contextual embeddings $h_{Ga}$ and $h'_{Ga}$. LayoutGMN is trained using the margin loss (margin=5) on the $L_2$ distances of the two paired embeddings.

between the two graphs, but requires additional computation. We use five rounds of message propagation, then the representation for each node is updated accordingly.

**Aggregator.** A 1024-D graph-level representation, $h_G$, is obtained via a feature aggregator MLP, $f_G$, that takes as input, the set of node representations $\{h^{(T)}_i\}$, as given below:

$$h_G = MLP_G \left( \sum_{i \in V} \sigma(MLP_{gate}(h^{(T)}_i)) \odot MLP(h^{(T)}_i) \right) \quad (4.7)$$

Graph-level embeddings for the two layout graphs is similarly computed.

$$h_{G_1} = f_G(\{h^{(T)}_i\}_{i \in V_1})$$

$$h_{G_2} = f_G(\{h^{(T)}_p\}_{p \in V_2})$$

### 4.3.3 Training

To learn a layout similarity metric, we borrow the Triplet training framework [45]. Specifically, given two pairs of layout graphs, i.e., anchor-positive and anchor-negative, each pair is passed through the same GMN module to get the graph embeddings in the context of the other graph, as shown in Fig 4.5. A margin loss based on the $L_2$ distance between the graph embeddings, as given in equation 4.8, is used to backpropagate the gradients through GMN.

$$L_{tri}(a, p, n) = \max(0, \gamma + \|h_{G_a} - h_{G_p}\|_2 - \|h'_{G_a} - h_{G_n}\|_2) \quad (4.8)$$
4.4 Datasets

We use two kinds of layout datasets in our experiments: (1) UI layouts from the RICO dataset [20], and (2) floorplans from the RPLAN dataset [151]. After some data filtering, the size of the two datasets is respectively, 66261 and 77669.

In the absence of a ground truth label set and the need for obtaining the triplets in a consistent manner, we resort to using IoU values of two layouts, represented as multi-channel images, to ascertain their closeness. Given an anchor layout, the threshold on IoU values to classify another layout as positive, from observations, is 0.6 for both UIs and floorplans. Negative examples are those that have a threshold value of at least 0.1 less than the positive ones, avoiding some incorrect 'negatives' during training. The train-test sizes for the aforementioned datasets are respectively: 7,700-1,588, 25,000-7,204. In the filtered floorplan training dataset [151], the distinct number of semantic categories/rooms across the dataset is nine and the maximum number of rooms per floorplan is eight. Similarly, for the filtered UI layout dataset [20], the number of distinct semantic categories is twenty-five and the number of elements per UI layout across the dataset is at most hundred.

4.5 Results and Evaluation

We evaluate LayoutGMN by comparing its retrieval results to those of several baselines, evaluated using human judgements. Similarity prediction by our network is efficient: taking 33 milliseconds per layout pair on a CPU. With our learning framework, we can efficiently retrieve multiple, sorted results by batching the database samples.

4.5.1 Baselines

Graph Kernel (GK) [28]. GK is one of the earliest structural similarity metrics, initially developed to compare indoor 3D scenes. We adopt it to 2D layouts of floorplans and UI designs. We input the same layout graphs to GK to get retrievals from the two databases, and use the best setting based on result quality/computation cost trade-off.

U-Net [117]. As one of the best segmentation networks, we use U-Net in a triplet network setting to auto-encode layout images. The input to the network is a multi-channel image with semantic segmentations. The network is trained on the same set of triplets as LayoutGMN until convergence.

IoU Metric. Given two multi-channel images, we use the IoU values between two layout images to get their IoU score, and use this score to sort the examples in the datasets to rank the retrievals for a given query.

GCN-CNN [93]. The state-of-the-art network for structural similarity on UI layouts is a hybrid network comprised of an attention-based GCN, similar to the gating mechanism
Figure 4.6: Top-5 retrieved results for an input query based on IoU metric, GCN-CNN_Triplet [93] and LayoutGMN. We observe that the ranked results returned by LayoutGMN are closer to the input query than the other two methods, although it was trained on triplets computed using the IoU metric. Attention weights for understanding structural correspondence in LayoutGMN are shown in Figure 5.1 and also provided in the Appendix (Section 4.7). UI and floorplan IDs from the RICO dataset [20] and RPLAN dataset [151], respectively, are indicated on top of each result. More results can be found in the Appendix (Section 4.7).

in [83], coupled with a CNN. In this original GCN-CNN, the training triplets are randomly sampled every epoch, leading to better training due to diverse training data. In our work, for a fair comparison over all the aforementioned networks, we sample a fixed set of triplets in every epoch of training. The GCN-CNN network is trained on the two datasets of our interest, using the same training data as ours.

Qualitative retrieval results for GCN-CNN, IoU metric and LayoutGMN for a given query are shown in Figure 4.6.

4.5.2 Evaluation Metrics

**Precision@k scores.** To validate the correctness of LayoutGMN as a tool for measuring layout similarity, we start by evaluating layout retrieval from a large database. A standard evaluation protocol for the relevance of ranked lists is the Precision@k scores [94], or $P@k$, for short. Given a query $q_i$ from the query set $Q = \{q_1,q_2,q_3,...,q_n\}$, we measure the
<table>
<thead>
<tr>
<th>Method</th>
<th>Precision@k (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k=1$ (↑)</td>
</tr>
<tr>
<td>Graph Kernel [28]</td>
<td>33.33</td>
</tr>
<tr>
<td>U-Net _Triplet [117]</td>
<td>27.08</td>
</tr>
<tr>
<td>IoU Metric</td>
<td>43.75</td>
</tr>
<tr>
<td>GCN-CNN _Triplet [93]</td>
<td>39.60</td>
</tr>
<tr>
<td>LayoutGMN</td>
<td><strong>47.91</strong></td>
</tr>
<tr>
<td>Graph Kernel [28]</td>
<td>27.27</td>
</tr>
<tr>
<td>U-Net _Triplet [117]</td>
<td>28.28</td>
</tr>
<tr>
<td>IoU Metric</td>
<td>33.84</td>
</tr>
<tr>
<td>GCN-CNN _Triplet [93]</td>
<td>37.37</td>
</tr>
<tr>
<td>LayoutGMN</td>
<td><strong>38.38</strong></td>
</tr>
</tbody>
</table>

Table 4.1: Precision scores for the top-k retrieved results obtained using different methods, on a set of randomly chosen UI and floorplan queries. The first set of five comparisons is for UI layouts, followed by floorplans.

The relevance of the ranked lists $L(q_i) = [l_{i1}, l_{i2}, ..., l_{ik}, ...]$ using the precision score,

$$P@k(Q, L) = \frac{1}{k|Q|} \sum_{q_i \in Q} \sum_{j=1}^{k} rel(L_{ij}, q_i), \quad (4.9)$$

where $rel(L_{ij}, q_i)$ is a binary indicator of the relevance of the returned element $L_{ij}$ for query $q_i$. In our evaluation, due to the lack of a labeled and exhaustive recommendation set for any query over the layout datasets employed, such a binary indicator is determined by human subjects.

Table 4.1 shows the $P@k$ scores for different networks described in Section 4.5.1 employed for the layout retrieval task. To get the precision scores, similar to [93], we conducted a crowd-sourced annotation study via Amazon Mechanical Turk (AMT) on the top-10 retrievals per query, for $N$ ($N = 50$ for UIs and 100 for floorplans) randomly chosen queries outside the training set. 10 turkers were asked to indicate the structural relevance of each of the top-10 results per query, without any specific instructions on what a structural comparison means. A result was considered relevant if at least 6 turkers agreed.

We observe that LayoutGMN better matches humans’ notion of structural similarity. [93] performs better than the IoU metric on floorplan data (+3.5%) on the top-1 retrievals and is comparable to IoU metric on top-5 and top-10 results. On UI layouts, the IoU metric is judged better by turkers than [93]. U-Net fails to retrieve structurally similar results as it overfits on the small amount of training data, and relies more on image pixels due to its convolutional structure. LayoutGMN outperforms other methods by at least 1% for all $k$, on both datasets. The precision scores on floorplans (bottom-set) are lower than on UI layouts perhaps because they are easier to compare owing to smaller set of semantic elements than UIs and turkers tend to focus more on the size and boundary of the floorplans in additional
to the structural arrangements. We believe that when a lot of semantics are present in the layouts and are scattered (as in UIs), the users tend to look at the overall structure instead of trying to match every single element owing to reduced attention-span, which likely explains higher scores for UIs.

**Overlap@k score.** We propose another measure to quantify the stability of retrieved results: the Overlap@k score, or Ov@k for short. The intuition behind Ov@k is to quantify the consistency of retrievals for any similarity metric, by checking the number of similarly retrieved results for a query and its top-1 result. The higher this score, the better the retrieval consistency, and thus, higher the retrieval stability. Specifically, if \( Q_1 \) is a set of queries and \( Q_1^{top1} \) the set of top-1 retrieved results for every query in \( Q_1 \), then

\[
Ov@k(Q_1, Q_1^{top1}) = \frac{1}{k |Q_1|} \sum_{q_m \in Q_1} \sum_{q_p = \text{top1}(q_m)}^k (L_{mj} \land L_{pj}),
\]

where \( L_{ij} \) is the \( j^{th} \) ranked result for the query \( q_i \), and \( \land \) is the logical AND. Thus, \( (L_{mj} \land L_{pj}) \) is 1 if the \( j^{th} \) result for query \( q_m \in Q_1 \) and query \( q_p = \text{top1}(Q_1) \in Q_1^{top1} \) are the same. \( Ov@k \) measures the ability of the layout similarity metric to replicate the distance field implied by a query by its top-ranked retrieved result. The score makes sense only when the ranked results returned by a layout similarity tool are deemed reasonable, as assessed by the \( P@k \) scores.

Table 4.2 shows the \( Ov@k \) scores with \( k = 5, 10 \) for IoU, GCN-CNN [93], and LayoutGMN on 50 such pairs. On UIs (first three rows), IoU metric has a slightly higher \( Ov@5 \) score (+0.6%) than LayoutGMN. Also, it shares the largest \( P@5 \) score with LayoutGMN, indicating that IoU metric has slightly better retrieval stability for the top-5 results. However, in the case of \( Ov@10 \), LayoutGMN has a higher score (+0.4%) than the IoU metric and also has a higher \( P@10 \) score than the other two methods, indicating that when top-10 retrievals are considered, LayoutGMN has slightly better consistency on the retrievals.
<table>
<thead>
<tr>
<th>2*Method</th>
<th>Test Accuracy on Triplets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IoU-based (↑)</td>
<td>User-based (↑)</td>
</tr>
<tr>
<td>Graph Kernel [28]</td>
<td>90.09</td>
<td>90.73</td>
</tr>
<tr>
<td>U-Net _Triplet [117]</td>
<td>96.67</td>
<td>93.38</td>
</tr>
<tr>
<td>GCN-CNN_Triplet [93]</td>
<td>96.45</td>
<td>94.48</td>
</tr>
<tr>
<td>LayoutGMN</td>
<td>98.96</td>
<td>95.80</td>
</tr>
<tr>
<td></td>
<td>92.07</td>
<td>95.60</td>
</tr>
<tr>
<td></td>
<td>93.01</td>
<td>91.00</td>
</tr>
<tr>
<td></td>
<td>92.50</td>
<td>91.80</td>
</tr>
<tr>
<td></td>
<td>97.54</td>
<td>97.60</td>
</tr>
</tbody>
</table>

Table 4.3: Classification accuracy on test triplets obtained using IoU metric (IoU-based) and annotated by users (User-based). The first set of comparisons is for UI layouts, followed by floorplans.

As for floorplans (last three rows), Table 4.1 already shows that LayoutGMN has the best $P@k$ scores. This, coupled with a higher $Ov@k$ scores, indicate that on floorplans, LayoutGMN has better retrieval stability. In the Appendix (Section 4.7), we show qualitative results on the stability of retrievals for the three methods.

**Classification accuracy.** We also measure the classification accuracy of test-triplets as a sanity check. However, such a measure alone is not a sufficient one for correctness of a similarity metric employed in information retrieval tasks [94]. We present it alongside $P@k$ and $Ov@k$ scores for a broader, informed evaluation, in Table 4.3. Since user annotations are expensive and time consuming (and hence the motivation to use IoU metric to get weak training labels), we only get user annotations on 452 triplets for both UIs and floorplans, and the last column of Table 4.3 reflects the accuracy on such triplets. LayoutGMN outperforms all the baselines by atleast 1.32%, on triplets obtained using both, IoU metric and user annotations.

4.5.3 Fully-connected vs. Adjacency Graphs

Following [93], we employed fully connected graphs for our experiments until now and observed that such graphs are a good design for training graph neural networks for learning structural similarity. We also performed experiments using adjacency graphs on GCN-CNN [93] and LayoutGMN, and observed that, for floorplans (where the graph node count is small), the quality of retrievals improved in the case of LayoutGMN, but degraded for GCN-CNN. This is mainly because GCN-CNN obtains independent graph embeddings for each input graph and when the graphs are built only on adjacency connections, some amount of global structural prior is lost. On the other hand, GMNs obtain better contextual embeddings by now matching the sparsely connected adjacency graphs, as a result of narrower search space; for a qualitative result using adjacency graphs, see Figure 4.7. However, for UIs (where the graph node count is large), the elements are scattered all over the layout,
Figure 4.7: Retrieval results for the bottom-left query in Fig 4.6, when adjacency graphs are used. We observe, on most of the queries, that the performance of LayoutGMN improves, but degrades in the case of GCN-CNN [93] on floorplan data.

and no one heuristic is able to capture adjacency relations perfectly. The quality of retrievals for both the networks degraded when using adjacency graphs on UIs. More results can be found in the Appendix (Section 4.7).

4.5.4 Ablation Studies on Structural Representation

To evaluate how the node and edge features in our layout representation contribute to network performance, we conduct an ablation study by gradually removing these features. Our design of the initial representation of the layout graphs (Sec 4.3.1) are well studied in prior works on layout generation [35, 80], visual reasoning, and relationship detection tasks [39, 165, 93]. As such, we focus on analyzing LayoutGMN’s behavior when strong structural priors viz., the edges, box positions, and element semantics, are ablated.

Graph edges. Removing graph edges results in loss of structural information, with only the attention-weighted node update (Eq. 4.4) taking place. When the number of graph nodes is small, e.g., for floorplans, edge removal does not lead to random retrievals, but the retrieved results are poorer compared to when edges are present; see Table 4.4.

Effect of box positions. The nodes of the layout graphs encode both the absolute box positions and the element semantics. When the position encoding information is withdrawn, arguably, the most important cue is lost. The resulting retrievals from such a poorly trained model, as seen in the second row of Table 4.4, are noisy as semantics alone do not provide enough structural priors.

Effect of node semantics. Next, when the box positions are preserved but the element semantics are not encoded, we observe that the network slowly begins to understand element comparison guided by the position info, but falls short of understanding the overall structure information, see Table 4.4. LayoutGMN takes into account all the above infor-
2*Structure encoding with

<table>
<thead>
<tr>
<th></th>
<th>Precision@k (%)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k=1 (↑)</td>
<td>k=5 (↑)</td>
<td>k=10 (↑)</td>
</tr>
<tr>
<td>No edges</td>
<td>30</td>
<td>16.39</td>
<td>11.3</td>
</tr>
<tr>
<td>No box positions</td>
<td>15</td>
<td>7.2</td>
<td>5.4</td>
</tr>
<tr>
<td>No node semantics</td>
<td>24</td>
<td>11.2</td>
<td>8.4</td>
</tr>
</tbody>
</table>

Table 4.4: Precision@K scores for ablation studies on structural encoding of floorplan graphs. The setup for crowd-sourced relevance judgements via AMT is the same as in Table 4.1, on the same set of 100 randomly chosen queries.

Figure 4.8: Element-level label transfer results from a source image $I_1$ to a target image $I_2$, using a pretrained LayoutGMN vs. maximum pixel-overlap matching. LayoutGMN predicts correct labels via attention-based element matching. Information returning structurally sound results (Table 4.1), even relative to the IoU metric. Qualitative results are provided in the Appendix (Section 4.7).

4.5.5 Attention-based Layout Label Transfer

We present layout label transfer, via attention-based structural element matching, as a natural application of LayoutGMN. Given a source layout image $I_1$ with known labels, the goal is to transfer the labels to a target layout $I_2$. A straightforward approach to establishing element correspondence is via maximum area/pixel-overlap matching for every element in $I_2$ with respect to all the elements in $I_1$. However, this scheme is highly sensitive to element positions within the two layouts. Moreover, raster-alignment (via translations) of layouts is non-trivial to formulate when the two layout images have different boundaries and structures. LayoutGMN, on the other hand, is robust to such boundary variations, and can be directly used to obtain element-level correspondences using the built-in attention mechanism that provides an attention score for every element-level match. Specifically, we use a pretrained LayoutGMN which is fed with two layout graphs, where the semantic encoding of all nodes is set to a vector of ones.

As shown in Figure 4.8, the pretrained LayoutGMN is able to find the correct labels despite masking the semantic information at the input. Note that when semantic information is masked at the input, such a transfer can not be applied to any two layouts. It is limited by a weak/floating alignment of $I_1$ and $I_2$, as seen in Figure 4.8.
4.6 Conclusion, limitation, and future work

We present the first deep neural network to offer both metric learning of structural layout similarity and structural matching between layout elements. Extensive experiments demonstrate that our metric best matches human judgement of structural similarity for both floorplans and UI designs, compared to all well-known baselines.

The main limitation of our current learning framework is the requirement for strong supervision, which justifies, in part, the use of the less-than-ideal IoU metric for network training. An interesting future direction is to combine few-shot or active learning with our GMN-based triplet network, e.g., by finding ways to obtain small sets of training triplets that are both informative and diverse [75]. Another limitation of our current network is that it does not learn hierarchical graph representations or structural matching, which would have been desirable when handling large graphs.

4.7 Appendix

**Attention Visualizations** LayoutGMN compares two layouts structurally via attention-based Graph Matching mechanism, in addition to message propagation within individual graphs. The former provides local structural correspondences, whereas the latter provides global structural prior for comparing two layouts. Specifically, if there exist $m$ semantic elements in layout $I_1$ and $n$ semantic elements in layout $I_2$, the attention-weight matrix for matching elements in $I_2$ w.r.t elements in $I_1$ is of size $n \times m$, and vice-versa. These attention weights change from layer-to-layer depending on the structural match. In Figure 4.9, we present two examples of floorplans with attention weights visualized in all 6 layers, with layer-0 being the layer where weights are initialized prior to training. For brevity, we just present floorplan attention visualization, and only show the largest attention weights, omitting all other (insignificant) connections.

**Fully Connected vs. Adjacency Graphs** All the quantitative results presented in Figure 4.6 are based on fully-connected graphs, for all the methods. We observed, both quantitatively and qualitatively (Fig 4.6, Table 4.1, 4.2, 4.3), that fully-connected graphs are a good input representation for learning structural similarity on layouts. We also experimented with adjacency graphs, on both, floorplans as well as UI layouts. As explained in Section 4.5, we observed that, for floorplans (where the graph node count is small), the quality of retrievals improved in the case of LayoutGMN, but degraded for GCN-CNN. A set of results for the same is shown in Figure 4.10. This is mainly because GCN-CNN obtains independent graph embeddings for each input graph and when the graphs are built only on adjacency connections, some amount of global structural prior is lost. On the other hand, GMNs obtain better contextual embeddings by now matching the sparsely connected adjacency graphs, as a result of narrower search space. However, for UIs (where the graph
Figure 4.9: Given a query (on the left) and its retrieved result (on the right), we show attention weights in different layers of message propagation, leading to element correspondences, from which structural similarity is driven and partly established. Layer-0 to Layer-5 show learned attention weights in different layers of propagation. For brevity, we only show the largest weights.
node count is large), the elements are scattered all over the layout, and no one heuristic is able to capture adjacency relations perfectly. The quality of retrievals for both networks degraded when using adjacency graphs on UI layouts; see Figure 4.11.

**Retrieval Stability**  In Section 4.5, we developed a new metric, called *Overlap@k* scores, to measure the stability of retrievals using different methods. This score measures the ability of the layout similarity metric to replicate the distance field implied by a query according to its top-ranked retrieval. Quantitative results for the same are shown in Table 2 in the main paper. In this manuscript, we present qualitative results for the same in Figure 4.12.
Figure 4.10: Additional retrieved results on floorplan queries, using adjacency graphs, and fully-connected graphs, using both, GCN-CNN [93] (left column), and LayoutGMN (right column). Note that all the quantitative results shown in the main paper are based on fully-connected graphs, following the design choice of [93].
Figure 4.11: Additional retrieved results on UI layout queries, using adjacency graphs, and fully-connected graphs, using both, GCN-CNN [93] (left column), and LayoutGMN (right column). Note that all the quantitative results shown in the main paper are based on fully-connected graphs, following the design choice of [93].
Figure 4.12: Retrieved results for a given query and its top ranked retrieval, using GCN-CNN [93] (left column) and LayoutGMN (right column). In every set of paired results (row-wise), the first row represents a query \( q \) and its top-5 retrievals. In the second row, the query is the top-1 result of query \( q \) in the first row, denoted by \( q^{top-1} \). Its top-5 retrievals shown against it.
Chapter 5

Recovering Shape Interiors from Sparse Articulation Images

5.1 Introduction

The emergence of large-scale 3D shape collections [11, 32, 99] has propelled the proliferation of neural 3D processing algorithms in computer graphics and vision. However, in the ensuing applications such as 3D recognition, reconstruction, and generation, the focuses of both the algorithms themselves and the evaluation have predominantly been placed on the (external) shapes and their visual appearance, largely overlooking the shape interiors. This is not surprising since the dominant majority of existing 3D models, e.g., those from ShapeNet [11], the largest and most frequently adopted training dataset, are only available as surface meshes. Even for 3D models carefully crafted by artists, e.g., from the Amazon-Berkeley Objects (ABO) [17] dataset, only the exterior surfaces were created.

With only an outside “shell”, a 3D object is incomplete as it is devoid of any of its potentially rich interior structures. Such models cannot adequately support common tasks closely tied to object motion and functionality, e.g., product inspection, physical simulation, and agent-object interactions in robotic and AR/VR. In particular, as a 3D object is interacted with and undergoes motions, its interiors may be revealed, e.g., the back planes of a drawer or the interior shelving of a cabinet when its door is opened; see Figure 5.1.

Our goal in this work is to recover the interior geometries of existing 3D models with only their exteriors from image guidance. To this end, we introduce a method to solve the new problem of multi-articulation recovery of shape interiors (RoSI). Specifically, given a set of RGB images that capture a target 3D object in different articulated poses, possibly from only few views, our method infers the interior planes that are observable from the input images. In our setting, we assume that the target 3D object is given as a “rest-state” mesh shell, i.e., a mesh without interior structures or part articulation. In addition to plane recovery, our method also predicts part articulations from the input images and is able
Figure 5.1: RoSI is a learning framework trained to recover the interior of a 3D model from articulation images which capture its various part articulations. We show results produced from three input RGB images (a) for a 3D cabinet provided in its rest (unarticulated) state; see (b)-left. As illustrated in (b)-right by an exploded view, the input 3D model has no interiors. Our method not only infers the interior planes, shown in (c) in solid colors, but can also reproduce, and extrapolate (e.g., door opening), the captured motions on the 3D model (d).
Figure 5.2: **Method pipeline.** Separate networks are trained for per-view camera pose estimation, depth estimation, and articulation detection in the first phase. The predicted poses and depths are fed into the second phase for per-view interior plane recovery, followed by multi-view fusion. The rest-state (i.e., unarticulated) 3D model is only used as input to the pose estimation network and optionally, for articulation detection.

To realize the captured motions on the target object in 3D, while exposing the predicted interior planes, as shown in Figure 5.1 for a cabinet.

It is well known that 3D geometry inference from sparse RGB images is a severely ill-posed problem, even when the outputs are restricted to planes. Recently, several learning-based methods have been proposed for plane detection from images, most notably PlaneR-CNN [86]. However, a unique challenge in our work comes from the input images, which capture an object undergoing different part articulations, with no constraints on the number of articulated parts in each image. In addition, the input images may exhibit significant diversity when they were captured from varying camera poses, both in terms of viewing distance and angle. These, along with the lack of image overlap due to view sparsity, would significantly compromise conventional multi-view 3D reconstruction approaches built on image correspondence. An additional implied difficulty arises when estimating camera poses, which are unknown but necessary to place the per-view predictions into a common coordinate frame, for plane integration and motion realization.

As outlined in Figure 5.2, our method constitutes a learning framework with two main phases. First, we perform *motion-aware multi-view analysis* which takes as input the multi-view, multi-articulation images and the rest-state, unarticulated 3D model, and predicts per-view camera poses, depth maps, and part articulations, using separate neural networks. Among these three tasks, camera pose prediction for the given rest-state 3D model from articulation images is particularly challenging due to the lack of correspondences, mainly over regions of articulated parts. To address this new challenge, we design a motion-aware neural architecture which simultaneously detects unarticulated image pixels and leverages them for feature matching, as detailed in Section 5.4. For articulation detection (Section 5.5),
we build upon OPD [57] and use the detected parts and associated motion parameters to transfer motions to 3D.

In the second phase, we recover shape interiors as planes in 3D (Section 5.6). Our network detects per-view 2D plane masks corresponding to the interior regions and infers their associated 3D plane normals in the camera coordinate frame. The interior planes in 3D space are then recovered based on the estimated camera pose, depths, per-view, and then merged in the common world frame via thresholding. Our neural model is built on PlaneRCNN [86], which was originally proposed for single-view plane detection in the camera frame. Although we process each articulation image independently, the key difference in our problem setting lies in projecting the view-recovered planes to a common coordinate frame for plausible interior shape recovery, which hinges critically on the pose estimation module. Putting these together, our contribution also includes proposing a system, which is the first one catered towards interior shape recovery from multi-articulation images.

All the modules in our method are trained using the PartNet-Mobility dataset [153] which provides \( \approx 2K \) 3D models with interior structures; the training is category-agnostic. Models are rendered using Mitsuba renderer [56], with an average of 19 view images per model (eventually, only 3 are considered), by random part articulations. Quantitative and qualitative evaluations show that our method achieves promising results by comparing against baselines and alternative solutions. We also perform tests on untrained object categories and on real images from the ABO dataset to assess the generalization capability of our approach.

5.2 Related work

In this section, we cover related works on 3D reconstruction, especially plane inference from images, neural 3D representation and reconstruction focusing on articulation, articulation detection from images, and pose estimation. We also discuss shape completion as a possible alternative for recovering 3D shape interiors.

3D reconstruction 3D reconstruction is one of the most studied problems in computer graphics [4]. In typical problem settings, a target 3D object is captured from one or more views in the form of RGB images, depth values, or point clouds, from which the 3D shape is to be recovered. While most 3D objects we encounter every day, e.g., common household items including furniture and appliances, exhibit various motions and articulations to perform their functions, conventional reconstruction methods assume that the target object maintains the same static and unarticulated pose in all views. This assumption is essential to a key step of the reconstruction, namely, multi-view registration or feature matching. At best, the resulting methods can only fully recover the object exteriors.
Recovering object interiors  To our knowledge, proactive scanning [162] is the only classical 3D reconstruction approach geared towards hidden object interiors. In this setting, human users actively adjust the target scene, e.g., opening a cabinet door or drawer, while continuously scanning it to reveal occluded regions. Despite its potential, this 3D acquisition paradigm introduces many technical challenges involving scan registration, human occlusion, as well as motion tracking and recognition. In clear contrast, our approach does not scan the object interiors; it relies on supervised learning to recover shape interiors from articulation images, while limited to what is visible. The rest-state 3D model is only available as a shell and its part articulations must be inferred from the images as well.

Shape completion is a possible alternative for synthesizing interior geometries, when the input consists of an exterior shell. While there have been many neural shape completion methods, mostly operating on point clouds [172, 105, 53] and with recent attempts utilizing transformers [183, 154, 163, 170], we are not aware of any work that specifically targets the recovery of shape interiors.

Neural reconstruction with articulation  Some recent works learn category-specific neural shape and appearance representations for articulated 3D shapes, e.g, NASAM [145] and A-SDF [100], where multi-view inference has been considered as an application. Most closely related to our work is StrobeNet [175], which reconstructs articulated 3D objects from multi-view and multi-articulation images. However, like the other works, StrobeNet also learns a category-level neural reconstruction model, i.e., it is trained per category and not dedicated to recover 3D shape interiors. In contrast, RoSI targets the specific problem of interior plane recovery and is trained in a category-agnostic manner.

Plane reconstruction from images  Early works [34, 36, 122, 123, 173] on planar surface reconstruction assign plane proposals over point clouds via global inference. In recent years, neural piece-wise planar reconstruction from a single image has gained quite a lot of interest, resulting in end-to-end architectures as proposed in [87, 86, 171, 131]. All these methods take one RGB image as input and output a depth image, plane segmentation masks and corresponding plane normals. 3D planar reconstruction is then made possible using the predicted depth map and camera intrinsic matrix. These methods all work on single-view images of indoor scenes, with no articulated objects. Our work deals with multi-view images of articulated 3D objects, where we detect interior planes from every view-image of the articulated model. And since the focus is to recover the interior geometry, obtaining a quality 2D reconstruction of interior depth maps is critical to the end solution. As such, we predict the depth maps separately using a transformer model [115], and build upon recent architectures [86, 114] to output interior planes from each articulated view image.

6DoF camera pose estimation  Standard pose estimation aims to predict the orientation and translation of an imaged object relative to a canonical CAD model. The majority
of prior methods assume that the image and the canonical model exhibit the same articulation state. To mitigate the performance drop due to object motions, various approaches have been dedicated to handle articulated objects, e.g., pose voting with random forest [97], articulation graph formulation with Markov Random Field [21, 109] from depth maps, pose transformation from optical flow on video inputs [89]. A number of works focus on category-level pose estimation by extending the Normalized Object Coordinate Space to normalize articulations into canonical motion states [81, 161, 146, 88]. The problem is also actively studied in Robotics where robotic arms are used for interaction [40, 64]. In this paper, we adopt a two-stage approach while proposing a motion-aware and category-agnostic neural architecture for simultaneous correspondence matching and articulation segmentation.

Articulation detection from images Akin to 2D object detection, the detection of articulated parts of furniture models from images aims at predicting the 2D part bounding box and part mask, as well as inferring various motion-related parameters on the articulated part such as motion type, motion axis, motion origin and motion magnitude. This finds utility in being able to transfer predicted motions from one articulated product image to many similar-looking static 3D models, thus helping create articulated 3D assets. To achieve this, a natural solution is to extend standard 2D object detection architectures, such as MaskRCNN [41]. Recent works from [57] and [114] build upon the MaskRCNN architecture, where the former focuses on detecting openable parts in articulated images and the latter doing so on articulation videos. Note that they do not focus on predicting motion magnitude. In our work, we build upon the work of [57] and detect all four motion parameters, in addition to 2D part bounding boxes and segmentation masks. We use these estimated 2D articulation parameters in our motion recovery step, as described in Section 5.5.1.

5.3 Overview

Figure 5.2 shows an overview of our system.
System input: The input to our system is a set of $N$ view-images, $\hat{I} = \{I_1, I_2, ..., I_N\}$, of an articulated 3D model, rendered from varying camera poses (both distance of the 3D model from the camera and the viewing angle). As well, each view-image captures varying articulations of the 3D model, in terms of the number of articulated parts and the magnitude of articulation per part. In addition, our system takes as input, the associated part-segmented (unlabeled), interior-devoid 3D model, $S$, in the rest state.
System output: With the above inputs, our goal is to: (a) recover interior structures revealed by images in $\hat{I}$, and (b) reproduce each input image in $\hat{I}$ in their associated articulations, in 3D.

As such, our problem tackles multiple tasks: learning to estimate pose of the 3D model $S$ per view, learning to estimate articulations from a single image, and learning to recon-
struct the interior geometry. In our work, we represent the interior geometry using plane as representation. We describe each task below.

5.4 Pose estimation from images

We propose a two-stage framework to estimate camera pose from articulation images as shown in Figure 5.3. Given multi-articulation RGB images and a rest-state 3D model, we first establish a pixel-to-point correspondence between every image and the 3D model defined in a global frame. To this end, a motion-aware neural architecture is proposed for simultaneously identifying unarticulated pixels (i.e., pixels belonging to unarticulated parts of the 3D model) and leveraging them for feature matching. Specifically, the RGB image (256×256×3) is passed into a U-Net [117] resulting in an image embedding of 256×256×32 dimensions. We also sample 2048 points from the rest-state 3D mesh, which are processed using DGCNN [144], resulting in a 2048×32 shape embedding. Instead of finding a hard mapping for each pixel, we generate a corresponding 3D point in a differentiable way. That is, we multiply the flattened image embedding with the shape embedding and apply a softmax function, resulting in a 256×2048 weight matrix. We then multiply it with the 3D point set and obtain the final “soft” points. Notice that such a soft point generation is also employed in point cloud registration [143].

Since object exhibits articulation in the captured image, feature matching is valid only at unarticulated pixels. To identify them, we multiply the weight matrix with the shape embedding, concatenate the multiplication result with the image embedding, and obtain a 256×256×64 tensor. We pass this tensor to a 2-layer CNN followed by a sigmoid function to output a binary segmentation mask. Our network is trained with L1 loss for the soft point generation branch and with the binary cross entropy loss for the segmentation branch. Finally, we compute the pose parameters using the unarticulated pixels with the Perspective-n-Point (PnP) algorithm.
5.5 Estimating articulations from images

Given a single RGB(D) image $I_j \in \{I_1, I_2, ..., I_N\}$, we aim to detect, in 2D, all the articulated parts $\{x_1, x_2, ..., x_k\}$ of the furniture model, and also predict, in 3D, the associated motion parameters $\{m_1, m_2, ..., m_k\}$. The output, thus, is a set of articulated parts and their associated motion parameters. Each part $x_i$ is represented by the set $\{label_i, bbox_i, mask_i\}$, where $label_i$ represents the part label (one of the three labels – lid, drawer, door), $bbox_i$ represents the 2D bounding box of the detected part, and $mask_i$ represents the 2D segmentation mask. Each motion parameter $m_i$ is represented by the set $\{t_i, o_i, a_i, v_i\}$, where $t_i$ represents the
motion type (revolute/prismatic), $o_i$ represents the motion origin ($3 \times 1$ vector), $a_i$ represents the motion axis ($3 \times 1$ vector) and $v_i$ represents the motion value/magnitude (scalar). For prismatic motions (e.g., pulling out a drawer), $m_i$ does not contain $o_i$ since motion origin is meaningful only for revolute motions (e.g., rotating a door around a hinge).

This problem draws parallels to the task of 2D object detection and instance segmentation, where each articulated part can now be thought of as an object instance. To this end, we build upon the Mask-RCNN architecture [41] which uses a Faster-RCNN style network for proposing bounding boxes on articulated parts, which are then processed using additional heads to output the instance label, 2D bounding box and 2D segmentation mask. Since we predict motion parameters for every detected articulated part, we add four independent branches at the output of RoIAlign layer to predict these motion parameters $m_i = \{t_i, o_i, a_i, v_i\}$, in the camera coordinate frame. This network is trained in a supervised manner, and the loss function used to optimize the network weights is given by:

$$l_{total} = (l_{label} + l_{bbox} + l_{mask}) + (l_{mtype} + l_{maxis} + l_{morigin} + l_{mmag}) \quad (5.1)$$

where the first three loss terms are the same as in Mask-RCNN [41], and the next four loss terms correspond to the four motion prediction heads. Specifically, $l_{label}$ is the negative log likelihood loss for part label, $l_{bbox}$ is the $L_1$ regression loss on the $4 \times 1$ vector representing
the 2D part bounding box, $l_{\text{mask}}$ is the average per-pixel sigmoid loss on the binary masks, $l_{\text{mtype}}$ is the binary cross-entropy loss for motion type, $l_{\text{maxis}}$, $l_{\text{morigin}}$ are the $L_1$ loss on the motion axis and motion origin, respectively, and $l_{\text{mmag}}$ is the MSE loss on the motion value. We do not make use of the predicted part labels (lid/drawer/door) in any of our modules (Figure 5.2). A loss against part labels is necessary for learning to localize articulated parts.

5.5.1 Motion realization in 3D

In order to transfer the predicted motions to 3D, we need to be able correspond an exterior part in the rest state segmented 3D model to a detected instance of part articulation. To do this, we predict the depth map corresponding to the input RGB image, as described in Section 5.6. Using the predicted camera pose and the articulation mask, we project the 2D depth image to 3D coordinate space, and reverse/undo the motion on the articulated part using the predicted motion parameters. We then find a one-way Chamfer Distance between each exterior mesh in the rest state segmented 3D model to the articulation-reversed 3D part pointcloud. The exterior mesh that gives us the least distance is then transformed from its rest state using the predicted motion parameters to achieve motion recovery, as shown in Figure 5.1 and 5.10.

5.6 Recovering interior geometry

5.6.1 Per-view plane detection

To recover interior geometry from the set of input images, we first detect, for every input image, all 2D plane masks corresponding to the interior regions and their associated 3D plane normals in the camera co-ordinate frame. Formally, given a single RGB image $I_j \in \{I_1, I_2, ..., I_N\}$, our goal is to predict, in 2D, all interior planes $\{p_1, p_2, ..., p_r\}$, and also predict, in 3D, the associated plane parameters $\{\pi_1, \pi_2, ..., \pi_n\}$. The output, thus, is a set of 2D interior plane masks and the associated planes in 3D. Each detected plane in 2D, $p_i$, is represented by the set $\{\text{label}_i, \text{bbox}_i, \text{mask}_i\}$, where $\text{label}_j$ represents the mask label (one of the two classes – planar/non-planar region), $\text{bbox}_j$ represents the 2D bounding box of the detected interior plane, and $\text{mask}_j$ represents the 2D segmentation mask of the plane. Each plane in 3D, $\pi_i$, corresponding to $p_i$, is parameterized by $[n_i, o_i]$, with $n_i$ being the plane normal, representing the coefficients $[a, b, c]$, and $o_i$ being the plane offset, such that $\pi_i^T[x, y, z, 1] = 0$.

Existing works [86, 59, 60] have shown the adeptness of R-CNN style networks in predicting plane representations. We follow suite. Similar to estimating articulations from images (Section 5.5), we employ Mask-RCNN architecture for the above task. So now, each planar region corresponding to the object’s interior is treated as an object instance (see Section 5.8.1 on how an object region is determined as interior). The plane prediction network then
detects such instances and estimates their segmentation masks and plane normals in the camera coordinate frame. The missing plane parameter, the plane offset $o_i$, corresponding to $\pi_i$, is obtained from the predicted depth maps, as explained below.

**Depth estimation.** To correctly recover the planes in 3D, it is essential to have a high-quality depth image corresponding to the input RGB image that would enable a not-so-noisy projection of the detected 2D plane masks to the 3D coordinate space. To this end, we fine-tune a transformer-based monocular depth estimation network [115] on our data, whose performance was observed to be better, particularly on the interior regions, compared to when the depth map was jointly predicted with the part masks and motion parameters.

**Plane offset estimation.** Recall that the plane normals are estimated in the camera coordinate frame. As such, we use the $3 \times 3$ camera intrinsic matrix, $K$, the predicted depth image and the estimated plane normals to obtain the offset as:

$$d = \frac{\sum_i m_i (n_i^T (z_i K^{-1} x_i))}{\sum_i m_i},$$

(5.2)

where $x_i$ is the $i_{th}$ pixel coordinate in the homogeneous representation, $z_i$ is its predicted depth value, and $m_i$ is a binary indicator for $x_i$ belonging to the 2D mask of plane $p_i$. 

Figure 5.6: Results on interior plane detection where each predicted plane is visualized using a different color. The top row shows the input image, middle shows predictions, and bottom row is for the ground truth.
5.6.2 Multi-view plane fusion

In this step, we combine predicted planes from multiple views by projecting from their camera coordinate frame to the world coordinate frame using predicted camera poses obtained from Section 5.4. Depending on the viewing angle, some number of planes coming from different view-images may overlap partially or almost completely, or maybe offset by small amounts. This indicates to the fact that they represent the same interior region. But since each image is processed independently, the relationship between them, i.e., inter-plane relation, is not encoded in any way to “stitch” them together.

As such, we merge such planes, in a pairwise manner, into a single plane if the plane normal difference between them is less than 15 degrees and if the smaller plane can be subsumed by the larger plane within a given threshold distance (which is 0.3 units in our case). Points from the smaller plane are projected onto the bigger plane if the above conditions are satisfied. Since a 3D shell is made available during testing, it can be used to regularize the position of the planes to fall within its boundaries. We visualize the planes as rectangles in our paper by using bounded values along the two plane axes.

5.7 Dataset

We use the PartNet-Mobility dataset [153] for all our experiments, which is a part-segmented 3D shape dataset with ground-truth annotation on part motions. We consider 9 categories of objects: Storage, Bin, Fridge, Microwave, Washer, Dishwasher, Oven, Safe and Box. When moveable parts in shapes belonging to these categories are articulated, they reveal planar interior regions, which is the reason such categories were selected to work with (as opposed to categories such as Stapler, Pliers, Scissors, USB etc.).

Since no annotations for object interiors are available, we identify interior regions by ray-casting. We shoot rays from a source and find faces on the part-segmented 3D object mesh the rays intersect with. An articulated 3D model reveals new intersecting faces that are not otherwise present when using the rest-state 3D model. These new faces are then said to correspond to the object’s interior. This setting allows us to obtain ground-truth depth, camera pose, binary articulated part masks (in 2D) and binary interior region masks (in 2D). To obtain ground-truth plane parameters (in 3D) for the interior planes, we use multi-plane RANSAC algorithm [25] on the 3D point cloud of 2D interior masks. Images are rendered using the Mitsuba renderer [56] at 256$^2$ resolution. We assume a consistent camera intrinsic matrix for all images. More details can be found in the supplementary. In total, after data filtering based on incorrect motion annotations and parts with few faces, we consider a total of 350 shapes, with a 75-25 train-test split.
Table 5.1: Quantitative evaluations of our motion-aware pose estimation network against PoseFromShape [156] and a variant of our method without the segmentation branch.

<table>
<thead>
<tr>
<th></th>
<th>Rotation error (°)</th>
<th>Translation error</th>
</tr>
</thead>
<tbody>
<tr>
<td>PoseFromShape</td>
<td>14.06</td>
<td>0.25</td>
</tr>
<tr>
<td>Ours (w/o segmentation)</td>
<td>9.53</td>
<td>0.13</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>7.61</strong></td>
<td><strong>0.10</strong></td>
</tr>
</tbody>
</table>

Table 5.2: Mean prediction errors on different motion parameters for RGB(D) image inputs from the test set: Raw network predictions vs. post-processing these raw predictions. R refers to Revolute motions and P for Prismatic. Only motion axis and origin are processed, since they are key to plausible motion realization in 3D.

<table>
<thead>
<tr>
<th>Error</th>
<th>Raw predictions</th>
<th>With post-processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(R)$ Axis angle(°) (↓)</td>
<td>16.58</td>
<td>14.5</td>
</tr>
<tr>
<td>$(R)$ Motion origin (↓)</td>
<td>3.69</td>
<td>0.64</td>
</tr>
<tr>
<td>$(P)$ Axis angle(°) (↓)</td>
<td>23.47</td>
<td>3.16</td>
</tr>
<tr>
<td>$(R)$ Magnitude(°) (↓)</td>
<td>28.6</td>
<td>-</td>
</tr>
<tr>
<td>$(P)$ Magnitude (↓)</td>
<td>0.165</td>
<td>-</td>
</tr>
</tbody>
</table>

5.8 Results and evaluation

We perform a module-wise evaluation of our system, providing insights on their individual strengths/weaknesses. In addition, we also quantitatively compare our interior recovery module against a baseline. Our pipeline is implemented in PyTorch, on a Tesla V-100 GPU. Figure 5.10 presents a gallery of results obtained from RoSI.

Evaluation metrics To evaluate the pose estimation module, we resort to the standard metrics of angular errors over predicted camera orientation and mean-squared error over translation, compared against the ground truth. We also measure the errors in predicted motion parameters, for both, revolute and prismatic motions, separately. As well, since our system eventually outputs interior planes in 3D, we compare its ability to recover such planes by using the following metrics: (a) measuring geometric similarity via a one-way Chamfer Distance (CD) to the ground truth (GT) planes, (b) mean normal angle difference in degrees, and (c) mean of the plane parameter differences with the ground truth plane.

5.8.1 Dataset

We use the PartNet-Mobility dataset [153] for all our experiments, which is a part-segmented 3D shape dataset with ground-truth annotation on part motions. We consider 9 categories of objects: Storage, Bin, Fridge, Microwave, Washer, Dishwasher, Oven, Safe and Box. When moveable parts in shapes belonging to these categories are articulated, they reveal planar interior regions, which is the reason such categories were selected to work with (as opposed to categories such as Stapler, Pliers, Scissors, USB etc.).
Table 5.3: We compare to a RANSAC-based baseline for plane fitting on the predicted depth maps for interior plane predictions and recovery, and to shape completion;

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Ours</th>
<th>Shape completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-way CD ((\downarrow))</td>
<td>0.0198</td>
<td>0.0137</td>
<td>0.151</td>
</tr>
<tr>
<td>Normal diff ((^\circ)) ((\downarrow))</td>
<td>22.421</td>
<td>5.700</td>
<td>-</td>
</tr>
<tr>
<td>Param diff ((\downarrow))</td>
<td>0.500</td>
<td>0.273</td>
<td>-</td>
</tr>
</tbody>
</table>

Since no annotations for object interiors are available, we identify interior regions by ray-casting. We shoot rays from a source and find faces on the part-segmented 3D object mesh the rays intersect with. An articulated 3D model reveals new intersecting faces that are not otherwise present when using the rest-state 3D model. These new faces are then said to correspond to the object’s interior. This setting allows us to obtain ground-truth depth, camera pose, binary articulated part masks (in 2D) and binary interior region masks (in 2D). To obtain ground-truth plane parameters (in 3D) for the interior planes, we use multi-plane RANSAC algorithm [25] on the 3D point cloud of 2D interior masks. Images are rendered using the Mitsuba renderer [56] at 256\(^2\) resolution. We assume a consistent camera intrinsic matrix for all images. More details can be found in the appendix (Section 5.10. In total, after data filtering based on incorrect motion annotations and parts with few faces, we consider a total of 350 shapes, with a 75%-25% train-test split.

5.8.2 Evaluation on motion predictions

Motion parameters in 3D allow us to recover motions depicted in images. We therefore evaluate motion predictions from the articulation detection network. We observe that the raw prediction results from the network are in the local neighborhood of the actual solution. As such, we post-process these predictions based on the observation that all models are axis-aligned in the canonical coordinate frame and that axes of motions align with the canonical axes. Do note that the starting solution for such post-processing is the network output. In Table 5.2, we compare the errors from these two methods. Motion origin error is the \(L_2\) loss between the ground truth and the prediction.

5.8.3 Evaluation on pose estimation

We compare our approach with another category-agnostic approach PoseFromShape [156], which directly regresses pose parameters with a neural network. Vanilla PoseFromShape predicts object orientation only, whereas we attach a linear layer to their network to estimate translation and retrain their network on our dataset. Table 5.1 shows that our approach outperforms PoseFromShape in both rotation error and translation error (about 2x better). Figure 5.4 shows some qualitative results where two viewpoints are provided in each example. The 3D bounding boxes computed using our poses are well aligned with the ground truth. Our method also accurately segments unarticulated regions compared
to the GT mask, providing reliable correspondences for the subsequent PnP step in our method. The ablation study in the last two rows in Table 5.1 validates the effectiveness of our unarticulated region segmentation.

5.8.4 Evaluation on interior recovery

To our knowledge, no prior works exist that perform interior recovery or articulated shape reconstruction from multi-articulation images. As such, we use the following baseline for comparison. Depth maps play a critical role in projecting plane masks to 3D space. GT depth maps can recover all interior planes when subject to multi-plane RANSAC plane fitting algorithm (and that’s how ground truth 2D plane masks were obtained for training plane detection network). We extend this idea on the predicted depth maps and fit planes over them, per-view basis, and compare against the ground truth. Parameters for the RANSAC plane estimation algorithm are: threshold distance between planes is set to $5e^{-3}$, and $\#\text{iters}$ at 2K.

Table 5.3 presents quantitative results of this comparison over three randomly chosen articulation view images per model and iterated hundred times to account for diverse view combinations. Our method outperforms the baseline on all three evaluation metrics. In particular, the geometry of the depth-associated 3D points does not allow for a meaningful plane estimation as seen via the normal angle difference and the mean of plane parameters difference. Since our method is explicitly trained to detect planes in 2D and their associated 3D normals, a better alignment to GT interior planes is expected.

Comparison to shape completion Finally, we investigate the hypothesis of using neural shape completion as an alternative to RoSI, with the important note that none of the current networks take image guidance as input. We chose the SOTA transformer-based shape completion network by [183] and train it on our dataset to allow for hallucinating interior points from exterior shell point clouds. Figure 5.7 shows qualitative results on examples where we could observe completed interiors. However, hallucination only from the exteriors clearly does not work well as much of the interior structures are still missing from the outputs; see also a comparison on one-sided CD (measured from GT planes since
Figure 5.8: RoSI results on untrained object category, table – two input images and corresponding recovered interior planes

shape completion results do not distinguish between interior and exterior points) in Table 5.3. While such a direction comparison may not be entirely fair, it serves to reinforce the importance of image guidance which characterizes our new problem setting for RoSI.

**Testing on untrained object category** We assess the generalization capability of our method by testing on models unused during training, specifically, the Table category. Figure 5.8 shows two such examples with their input images. Not only can RoSI recover interior planes, but it can also properly reproduce part motions in 3D.

**5.9 Conclusion, limitation, and future work**

Our ultimate goal is to endow all existing 3D models with interior structures to truly complete them. To this end, we present RoSI, a learning framework to recover 3D shape interiors from multiple, sparse views of object articulation images. To the best of our knowledge, this is a new problem, while our current solution only represents a preliminary attempt which is still limited in several ways.

First, we can only recover, in 3D, those interior planes that are visible in at least one of the input images. RoSI currently lacks the ability to learn to reconstruct non-observable interior structures. Simple heuristics resorting to plane extension or symmetry-based structure completion may be employed. On the other hand, with training 3D shapes possessing full interiors, it is possible to re-formulate the learning problem to allow hallucinating unseen interior regions from their exterior contexts or other priors.

Second, our articulation/plane detection is performed in a per-view fashion and we use the predicted camera poses to project the inferred planes into a common frame. In doing so, no view consistency is enforced in learning the planes in 3D. While our design choice is motivated in part by the lack of image overlaps due to view sparsity, view consistency, if learned in the plane normal estimation step, could remove the dependency on a separate pose estimation network.

Third, we discovered a clear domain gap when testing on real articulation images, from the ABO dataset [17]. Our method has been trained with synthetic data from PartNet-
Mobility which lacks the realism and diversity in terms of textures, lighting, material, etc. Our pre-trained plane detection model fails to robustly detect most planes on these real images; see Figure 5.9. Such a domain gap could be addressed via data augmentation.

Last but not least, while there is an abundance of online images showing part articulations, the dependency on having a quality 3D mesh shell during testing may not be practical. Foregoing this dependency would open up opportunities to develop neural architectures that can simultaneously reconstruct the exterior surface and generate interior structures from RGB images, particularly with plane representation, giving us means to create diverse and functional 3D assets. In addition, most articulation images of online products come with accessories which can compromise our current plane detection scheme. This may be remedied with data augmentation.

For future work, besides addressing the limitations discussed above, we may also analyze inter-plane, and more generally, structural relations between object parts, across views, as a means to improve both plane inference and integration; designing a learning scheme for such relations is a novel problem in itself. At last, generalizing our entire pipeline to accommodate more general 3D primitives and object types, are also necessary to fulfill our ultimate goal.

5.10 Appendix

Dataset  We use the PartNet-Mobility dataset [153] for all our experiments. We render a 3D model from out of the nine categories from 19 viewing angles, where the camera is placed at random viewing angles in the $[-\pi, \pi]$ range and at random distances from the object. We capture articulations on the object using a Scene Rendering Toolkit, as employed in OPD [57]. In doing so, we articulate random number of parts per object in random magnitudes, which gives us diversity of object models (and therefore the view rendered images).

We identify object interiors by shooting rays from a source and identifying faces on the part-segmented 3D object mesh the rays intersect with. This setting allows us to obtain all the metadata needed for our pipeline, as demonstrated in Figure 5.11.
Figure 5.10: Qualitative results on interior recovery from three multi-articulation images. The first three columns show the input, the next two columns show the recovered planes visible in the input images, overlayed on the input 3D shell (semi-transparent) corresponding to the input images. The next three columns show the articulated 3D models with interior structures, rendered from the same viewing angle as the input images. Articulations are realized in 3D based on part detection and motion predictions from individual images.
Figure 5.11: Left to right – RGB Art (Articulated RGB image), Art mask (articulated object mask), Ori image (Rest state object image), Ori mask (rest state object mask), Arti area mask (mask of articulated parts of the 3D object), Inte area mask (mask of the interior regions of the 3D model). Given a 3D model, we shoot rays to its surface, both in the rest state and the articulated state. This is done to identify interior regions by finding newly intersecting faces compared to intersecting faces when rays hit the rest state 3D model. This setting allows us to get all of the metadata needed for all the modules presented in Figure 2 of the main paper.
Estimating articulation from images  In Figures 5.12, 5.13, we show results on articulation detection and depth estimation from a single-view RGB image.

In general, the predictions on the motion parameters, when the articulation mask and part label are correctly detected, lie in the local neighborhood of the actual solution. This is not usable if we want to realize the motions in 3D. As such, we process these results by simple thresholding – if the motion axis is within 20 degrees of alignment to any of the axes, we consider that particular axis as the new motion axis. Based on the new motion axis, the motion origin (in case of Revolute motions) is accordingly modified since the rest state exterior part is known to us. Note that this processing is performed under the assumption that all models are axis aligned and that the motion on individual parts in the PartNet-Mobility dataset are along one of the three axis.
Figure 5.12: Left to right – Input RGB image, GT articulation masks and motion parameters, unarticulated GT depth map, GT interior depth map, predicted interior depth map and depth map difference.
Figure 5.13: Left to right – Input RGB image, GT articulation masks and motion parameters, unarticulated GT depth map, GT interior depth map, predicted interior depth map and depth map difference.
Chapter 6

Conclusions and Future Directions

Learning to model indoor scenes along with an understanding of the interactability of the constituent 3D objects is essential for a functional understanding of indoor environments, which can enable the development of advanced algorithms for designing interior spaces. This thesis explores the design of 3D indoor scenes, with advanced algorithmic development and evaluation in mind, going from the scene layout level to the object level, where functionality plays a key role in both cases. At the scene layout level, a generation paradigm that learns object occurrence patterns from a dataset of synthetically created scenes where object arrangements conform to functional usage of indoor space, is presented. Such an approach leads to the synthesis of novel 3D indoor scenes. An interesting problem follows, one of comparing and evaluating the generating 3D scenes, for which a learning-based algorithm is lacking. We present a work that makes a preliminary attempt in this direction by first considering abstractions of 3D indoor scenes, which are nothing but 2D floorplans, and presents a neural graph matching technique to measure the structural similarity of such 2D layouts. Finally, for understanding and modeling functionality at the object level, a reconstruction paradigm is presented, where a neural framework for recovering 3D shape interiors and realizing 3D part motions from sparse multi-articulation images of 3D shapes is presented.

We regard our works as the first step in their respective directions. In this chapter, we conclude the thesis with a summary of the main contributions of our methods, followed by possible directions for future research.

6.1 Conclusions

In this thesis, we presented three works: GRAINS (Chapter 3), LayoutGMN (Chapter 4), and RoSI (Chapter 5). All of these works leverage the representational power of neural networks to address the respective tasks. We summarize the key takeaways from these works below. Limitations of these works have been described in Chapters 3.7, 4.6, and 5.9, respectively, and therefore, not covered here for avoiding redundancy.
GRAINS makes a first attempt at developing a generative neural network to learn hierarchical structures of 3D indoor scenes. The key insight here is that indoor scenes are inherently hierarchical, and as such, can be represented using hierarchies based on object occurrence patterns. Leveraging hierarchical structure, GRAINS learns scene patterns in the form of commonly occurring sub-scenes at different scales. To this end, it integrates a recursive neural network with a variational autoencoder, enabling us to generate a novel, plausible 3D scene from a random vector in less than a second. The network design consists of several unique elements catered to learning scene structures for rectangular rooms, e.g., relative object positioning, encoding of object semantics, and use of wall objects as initial references, distinguishing itself from previous generative networks for 3D shapes [150, 79].

LayoutGMN develops the first deep neural network for learning structural similarity of floorplans (and 2D layouts in general), which offers both metric learning of layout similarity and structural matching between layout elements. The main insight of this work is that in order to learn reliable structure embeddings for comparing room arrangements in two floorplan layouts, their embeddings should be learned in a contextual manner, rather than learning independently. That is, the embeddings of one floorplan should be dependent on the structural arrangement of rooms in the other floorplan it is being compared with. This is made possible by the use of an attention-guided graph matching mechanism. Effective embeddings are learned due to the use of contrastive learning setting in the form of a triplet network that can separate similar floorplan embeddings from dissimilar ones.

RoSI is a category-agnostic learning framework trained to recover 3D shape interiors, and to the best of our knowledge, this is a new problem, introduced and addressed for the first time. Given a set of RGB images that capture a target 3D furniture model in different articulated poses, possibly from only few views, RoSI infers the interior planes that are observable in the input images. This is made possible by a motion-aware multi-view analysis phase including pose, depth, and motion estimations, followed by interior plane detection in images and 3D space, and finally multi-view plane fusion. In addition, RoSI also predicts part articulations and is able to realize and even extrapolate the captured motions on the target 3D object.

6.2 Future Research Directions

Recall the family conversation from Chapter 1. The eventual goal being advocated for by this thesis is the development of computational tools for assisting interior design tasks, by building a system, as the wife envisioned, “that could automatically recommend different
designs for different room types, while also allowing interactions with the placed furniture items to understand how they function and how their interiors would look". In short, this is to say that the modeling of indoor scenes should not be divorced from the functional understanding of the environment, as well as their constituent 3D objects as they partly determine scene usage. While it is not straightforward to incorporate the two in a learning framework, the first direction below provides helpful pointers in the pursuit of this specific goal, whereas the subsequent directions discuss other kinds of immersive experiences that could be integrated into the functional modeling of indoor environments.

**Modeling scene interactions.** For an immersive experience in the virtual world (the so-called metaverse), users should be able to interact with objects in a scene. This can occur in two ways – (1) using objects and furniture in their intended purposes (ex: sitting on a chair, adjusting the sofa to appropriately face the tv, using the tv remote, picking up a pen), and (2) playing with furniture models (ex: facilitating articulations on the drawers of a file cabinet, where the user can touch a drawer and it pops open; allowing part mobility such as adjusting the height of a chair seat; manipulating object functionalities in novel ways, perhaps making them non-functional). Such data observations, in the form of subscene-level (or even single object) human-object interaction videos or photographs, can be leveraged to extract interaction points on 3D objects, which I would like to call, 3D hotspots (or HOIs-pots). Such 3D hotspots, along with the human activity/movement regions, can be utilized in designing liveable indoor scenes, where 3D hotspots are correlated to articulate-able parts of the furniture items, either directly through observed data or through other priors, such as symmetry in 3D or similarity of region features in 2D. With a functional understanding of the 3D objects, scene generation algorithm can be specifically modeled to use such interaction priors to select objects when populating/instantiating a scene, which in turn, will dictate the placement of other items to complete the scene. It is worthwhile to point out prior attempts [121, 77, 51, 49, 48, 50] that have focused on modeling human-object interactions in the context of object functionality and mobility. However, to our knowledge, leveraging scene interactions on interactable objects for scene understanding and generation is not explored and is a direction worth investigating.

**Modeling dependencies between scenes and human motions.** Another major factor that affects object arrangements in a scene is human activity and motion sequence. In fact, one of the major factors taken into account when designing indoor environments is an unconstrained and obstacle-free space to support human activity. As a result, there exist strong relational priors between human activity and scene layout. One can leverage these priors to predict object arrangements based on human motion or vice-versa, i.e., predict human motion sequences based on either a subscene or a full scene. For example, a sitting human pose usually implies the presence of a chair or a sofa underneath, and the posture of
human legs can further reveal information about the base structure (legs) of the supportive surface, be it a chair or a sofa. This is somewhat related to modeling scene interactions, but now, the idea is to look at interactions at the object level instead of at the part level as otherwise discussed in the above paragraph. Recent works such as [166, 167] have explored this line of research, wherein they learn to predict object placements, including the object category, orientation, and object extent by learning from a database of human motion sequences in indoor scenes. Likewise, the composition of a sub-scene can throw light on the possible set of actions a human may perform, which in turn, can inform about the end state human pose for each of these action states. This can prove useful in synthetically creating (or augmenting) human motion and pose datasets for indoor environments, which are otherwise expensive to capture. Developing a learning framework for the latter task can be extremely useful for home-buying experiences in the context of VR/AR platforms. Overall, learning to model the dependencies between object arrangements and human motions can potentially generate truly functional indoor spaces for all kinds of conceivable human motions.

**Domain translation for learning part articulations** Another modeling paradigm for articulated objects is to develop either an image-to-image or a shape-to-shape translation algorithm that can seamlessly transition between one instance of a rest-state object and its articulated instance (for ex., a cabinet with a drawer closed and a cabinet with a drawer opened). This is especially important considering the fact that the number of furniture models in the PartNet-Mobility dataset [153] (the largest dataset of articulate-able furniture models) is quite small, and it takes a significant amount of effort in creating such datasets. A major challenge in this task is to decide (or learn) how many parts should be considered in the articulation state. As a potential first step, it is a good idea to start with just one part and accomplish this successfully. Investigations should also be led into achieving shape translations on articulated objects, across instances of different object categories. This is a highly challenging task, but one can imagine why this can be quite fruitful for content creation as well as understanding. Finally, it is worthwhile to investigate the possibility of this technique to generate instances of shape interiors, which can present an interesting application of employing domain translation for hallucinating or generating shape interiors.

**Modeling geometry and object textures for scene synthesis.** Access to a rich 3D scene database that contains diverse 3D objects is crucial for building neural models tasked with scene generation. In all such works, objects are placed in the generated scene layout by retrieving from an object database based on generated object attributes, which typically are nothing but bounding box dimensions, category, and centroid. Although geometric information is generated, appearance properties are not accounted for. This needs to be addressed since existing structural and geometric attributes provide strong cues to material and tex-
ture appearance, something that has been under-explored at the object level [55, 85, 15]. Learning to model a coupling of this with the scene layouts is an interesting approach to bypass the typical object retrieval step and directly generate objects with novel appearances.

**Move planning, scene rearrangement, and teleportation.** The ability to move freely and efficiently in indoor environments influences productivity and work culture, either for shared workplaces such as offices, restaurant kitchens, warehouses, and airports, or for personal spaces such as living rooms and open kitchens. More often than not, such planning takes ample time and layout considerations, especially with a large object inventory. Suggesting a plausible arrangement of objects leading to an optimized workplan, as well as space design is extremely useful in industrial applications. In a recent work [176], such workspaces and workplans are automatically designed given the input space and workspace equipment, in addition to staff properties as inputs. Such designs may benefit from data-driven modeling, for which, a diverse, rich, large-scale database of indoor environments spanning different industries is the first necessary step.

This move planning concept can be used to inject teleportation features in the metaverse, where a system can automatically suggest possible teleportation locations within an already-seen environment based on navigability, and the user can hop between such spots virtually. One way to do this is to sample a set of desirable teleport positions, assessing ease of navigation (and properties such as coverage and connectivity from a subscene or a focal point [157]). Such an application has recently been explored in [76] which synthesizes scene-aware teleportation graphs.

A slight deviation from the above, but with the potential to serve teleportation application, albeit slower, would be to allow the user to select teleportation spots apriori in an already configured environment, and develop a system that can re-arrange objects in the current physical state of the environment to a new state so as to optimize for the desired tele-movement. Different versions of this task have been described in [3] where the target environment state can be described by object poses, images, language description, or by letting an agent experience the target state environment, if possible. A recent work [137] makes use of reinforcement learning for automatic move planning of 3D objects from an initial 3D layout to a target layout. This application serves well in practice, and requires further exploration leveraging recent score-based models (a.k.a diffusion models) [126, 44].
Bibliography


