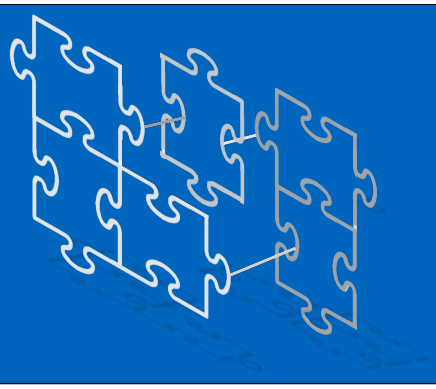
Computer Aided Medical Procedures
Prof. Dr. Nassir Navab



Dissertation

Scene Understanding and Decomposition for Synthesis and Editing

Helisa Dhamo

Fakultät für Informatik
Technische Universität München

Technische Universität München
Fakultät für Informatik

Scene Understanding and Decomposition for Synthesis and Editing

Helisa Dhamo

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung einer

Doktorin der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende(r):	Prof. Dr. Rüdiger Westermann
Prüfer der Dissertation:	1. Priv.-Doz. Dr. Federico Tombari 2. Prof. Dr. Stefan Leutenegger 3. Prof. Gregory D. Hager

Die Dissertation wurde am 27.09.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 19.01.2022 angenommen.

Helisa Dhamo
Scene Understanding and Decomposition for Synthesis and Editing
Dissertation, Version 1.0

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Informatikanwendungen in der Medizin
Boltzmannstraße 3
85748 and Garching bei München

Abstract

Scene synthesis consists in generating data that represents a scene, often in compliance with the constraints imposed by a given condition. Scene editing is a form of synthesis in which a given scene is *partially* generated, as a result of removing, adding or changing certain parts. View synthesis on the other hand, given a set of images of a scene from different views, consists in generating an image of the scene from an unseen view. This can be thought of as a form of image editing, where the content remains the same, while the view is changed.

In this dissertation, we explore deep learning approaches to tackle the challenge of scene generation in its full or partial form. In particular, we leverage different forms of compositional representation to decompose the scene in a set of known parts. We build these representations by *understanding the scene*, *i.e.* extracting meaningful geometric and semantic information, such as depth, object class labels and extent, as well as interaction between objects.

First, we investigate *layered depth images (LDI)* to enhance the representation of a single photo. This representation contains multiple texture and depth values per pixel, including knowledge on the occluded pixels, which makes it suitable for view synthesis. The task of synthesizing a layered depth image is thus generation of occluded parts, provided the visible parts. We explore a semantic extension of an LDI, and demonstrate its superior performance compared to the baselines, while enabling an additional object removal task.

Further, we explore a semantic representation based on *scene graphs*, where nodes are scene entities (objects) while the edges denote inter-object relationships. Heretofore, scene graphs have shown to be useful for many tasks, including conditional image generation and retrieval. Instead, we propose the first work that leverages scene graphs for image manipulation, where a user can make changes in an image by simply modifying the nodes and edges of a graph. To enable changing of certain image parts, while preserving the remaining content, we enrich the vanilla scene graph representation with neural visual features and bounding boxes obtained per scene entity.

Finally, we want to explore a similar scene graph representation for 3D scene generation. To circumvent the lack of appropriate data, we first create a large-scale dataset annotated with 3D semantic scene graphs and introduce the first learned method that can predict a scene graph from a 3D scene. We further use the acquired data to train a model for simultaneous generation and manipulation of 3D scenes using scene graphs, where we extend the graphs with 3D bounding boxes and latent shape codes. Thereby, we demonstrate encouraging results in the joint learning of boxes and shapes, which stimulates further research in fully-learned scene generation models.

Zusammenfassung

Die Szenen Synthese besteht in der Erzeugung von Daten, die eine Szene darstellen. Dabei werden oft verschiedene Einschränkungen auferlegt, die dabei einzuhalten sind. Szenen Bearbeitung ist eine Form der Synthese, bei der eine gegebene Szene teilweise generiert wird, indem bestimmte Teile entfernt, hinzugefügt oder verändert werden. Bei der Synthese von neuen Ansichten wird hingegen ein Bild der Szene aus einem ungesehenen Blickwinkel, mithilfe einer gegebenen Reihe von Bildern der Szene, erzeugt. Dies kann als eine Form der Bildbearbeitung angesehen werden, bei der der Inhalt gleich bleibt, während nur die Ansicht geändert wird.

In dieser Dissertation untersuchen wir Deep-Learning-Ansätze, um die Herausforderung der Szene Generierung in ihrer vollständigen oder teilweisen Form anzugehen. Insbesondere nutzen wir verschiedene Formen von kompositorischen Repräsentationen, um die Szene in eine Reihe bekannter Teile zu zerlegen. Wir bauen diese Darstellungen auf, indem wir zuerst versuchen die Szene zu verstehen, d. h. aussagekräftige geometrische und semantische Informationen zu extrahieren, wie z. B. Tiefe, Objektklassen und Ausdehnung, sowie die Interaktion zwischen Objekten.

Zunächst untersuchen wir geschichtete Tiefenbilder (LDI), um die Darstellung eines einzelnen Fotos zu verbessern. Dabei enthält diese Repräsentation mehrere Textur- und Tiefenwerte pro Pixel, einschließlich des Wissens über verdeckte Pixel. Dies macht diese Darstellungen besonders für die Synthese von Ansichten sehr geeignet. Die Aufgabe der Synthese eines geschichteten Tiefenbildes besteht also darin, die fehlenden Teile zu generieren die von anderen Objekten oder Teilen verdeckt sind. Wir erforschen eine semantische Erweiterung diese LDI Darstellung und demonstrieren ihre Überlegenheit verglichen mit den Basismodellen. Außerdem bietet diese Darstellung zusätzlich die Möglichkeit Objekte zu entfernen.

Darüber hinaus erforschen wir eine weitere semantische Repräsentation auf der Grundlage von Szenengraphen, bei denen die Knoten Szenen Objekte darstellen, während die Kanten die Beziehungen zwischen den Objekten symbolisieren. Szenengraphen haben sich bereits als nützlich für viele Aufgaben erwiesen, einschließlich der bedingten Bilderzeugung und -suche. Im Gegenzug schlagen wir die erste Arbeit vor, die Szenengraphen zur Bildmanipulation verwendet, bei der ein Benutzer Änderungen an einem Bild vornehmen kann, indem er einfach die Knoten und Kanten eines Graphen verändert. Damit bestimmte Teile des Bildes verändert werden können, während der restliche Inhalt erhalten bleibt, reichern wir die Standard-Darstellung des Szenengraphen mit neuronalen visuellen Eigenschaften und Bounding Boxes an, die für jedes Szenen Objekt erhalten werden.

Schlussendlich wollen wir eine ähnliche Szenengraph-Darstellung für die Erzeugung von 3D-Szenen erforschen. Um den Mangel an geeigneten Daten zu umgehen, erstellen wir zunächst einen großen Datensatz, der mit semantischen 3D-Szenengraphen annotiert ist. Des Weiteren, stellen wir die erste erlernte Methode vor, die einen Szenengraphen aus einer 3D-Szene bestimmen kann. Die gewonnenen Daten werden auch verwendet, um ein Modell für die gleichzeitige Erzeugung und Manipulation von 3D-Szenen mit Hilfe von Szenengraphen zu trainieren, wobei wir die Graphen mit 3D-Bounding-Boxen und latenten Codes zur Beschreibung der Körpereigenschaften erweitern. Dabei zeigen wir ermutigende Ergebnisse beim gemeinsamen Lernen von 3D Bounding Boxes und Körpermerkmalen, was zu weiteren Forschungen über vollständig gelernte Modelle zur Szene Generierung anregt.

Acknowledgments

Disclaimer: This PhD has been completed without a single drop of coffee intake. Yes. I thank all people who fueled me with their positive thoughts, support, food, art, music, and good ideas instead.

It has been a special pleasure working with Federico Tombari, a great advisor not only for the PhD, but also for life. Special thanks also goes to Nassir Navab, for all the fruitful discussions and for creating an amazing group of people. I am very grateful to both for trusting me with this opportunity in the first place and supporting me on the way. I would also like to thank Gregory Hager for the inspiring discussions during our collaboration.

I want to express my gratitude to all my amazing colleagues: Anees, Ashkan, Azade, Benjamin, Christian, David, Evin, Hendrik, Huseyin, HyunJun, Iro, Keisuke, Johanna, Mahdi, Manuel, Maria, Mira, Niko, Shun-Cheng, Tolga, Yanyan and Yida. Thanks for the discussions, the co-sufferings, the travels, the delicious food gatherings and the music playing nights. A special mention goes for the most senior ones, Iro, Christian, Keisuke and David for guiding me from the start, providing their useful tips and tricks and help me solidify my ideas. I also thank Martina Hilla for her magic, without which the chair would not be the same.

I am particularly thankful for my internship at Facebook and the collaboration with Lei Xiao and Feng Liu. I enjoyed our discussions and I learned a lot in those three months.

I am very grateful to my family and friends for their love and support. My mom, for showing me through her example that hard work never goes to waste. My dad, for involving me in his engineering projects at home since I was little, (and for proudly advertising my work on social media). My sister Livia, for the nice drawings on my office whiteboard. My friends, Desi, Arba, Emil, Alberto, Su, Saumitra, Erti and Aida who have always been there for me, and cheered me up during these PhD years. I'm lucky that I shared this journey with Fabian, my colleague slash boyfriend, who has always been understanding about my late working hours – because he was in the office next door fighting his own battles. Thanks Fabian for your love, ideas, and for encouraging me to keep going in the most difficult moments.

Last but not least, I want to thank whoever invented the big slide at the Informatics Faculty of TUM. It has been a great way to replace stress with adrenaline after each paper submission.

Contents

Chronological List of Authored and Co-authored Publications

2021

[20] **H. Dhamo***, F. Manhardt*, N. Navab, F. Tombari. "Graph-to-3D: End-to-end Generation and Manipulation of 3D Scenes Using Scene Graphs". *IEEE International Conference on Computer Vision (ICCV).* 2021

[33] S. Garg*, **H. Dhamo***, A. Farshad, S. Musatian, N. Navab, F. Tombari. "Unconditional Scene Graph Generation". *IEEE International Conference on Computer Vision (ICCV).* 2021

2020

[19] **H. Dhamo***, A. Farshad*, I. Laina, N. Navab, G. D. Hager, F. Tombari, C. Rupprecht. "Semantic Image Manipulation using Scene Graphs". *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2020

[141] J. Wald*, **H. Dhamo***, N. Navab, and F. Tombari. Learning 3D Semantic Scene Graphs from 3D Indoor Reconstructions". *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2020

2019

[21] **H. Dhamo**, N. Navab, F. Tombari. "Object-Driven Multi-Layer Scene Decomposition from a Single Image". In: *IEEE International Conference on Computer Vision (ICCV).* 2019

[22] **H. Dhamo**, K. Tateno, I. Laina, N. Navab, and F. Tombari. "Peeking Behind Objects: Layered Depth Prediction from a Single Image". *Journal Pattern Recognition Letters (PRL).* 2019

* The first two authors contributed equally

Part I

Introduction

1 Introduction

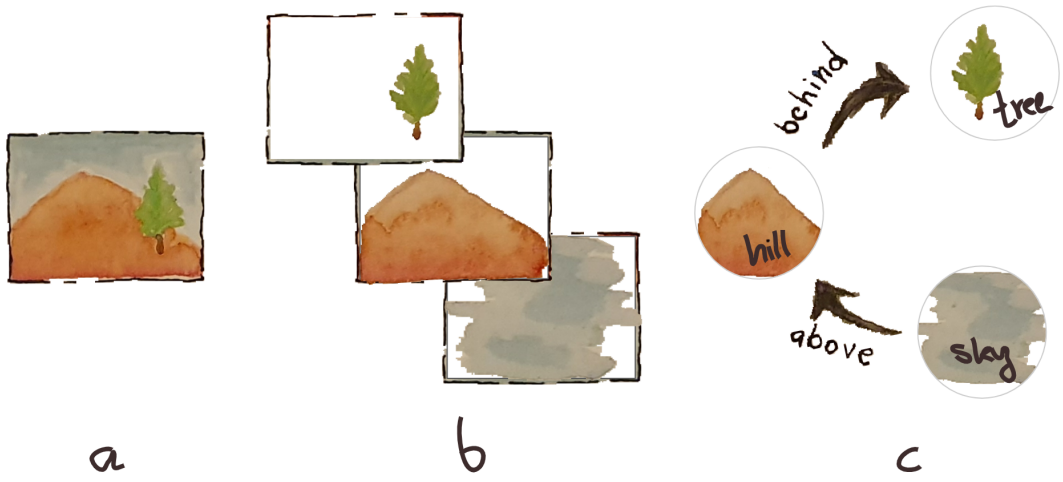


Figure 1.1. Compositional representations for scenes. a) exemplary scene, b) layered depth representation, c) scene graph.

1.1 Motivation and Main Objective

Automatic *scene synthesis* consists on generating new scenes, which have not been perceived before by the underlying algorithm. This can be carried our either conditionally, *i.e.* based on an input that constrains the output space as desired; or unconditionally, leading to a random outcome. It is of high interest in many fields, including augmented and virtual reality (AR/VR), facilitating the work of artists and designers, as well as modeling and understanding the world. This is a highly challenging field in computer vision, as it is important to assure that the generated scenes represent a potential real world scenario. Therefore, a model that synthesizes scenes, should be capable of first analysing and *understanding* the real world and the constraints therein. This involves, for instance, human action, object co-occurrence, and inter-object interactions.

Scene *manipulation* or *editing* is a variant of scene synthesis, which instead of generating the entire content, focuses on changing some aspects of an existing scene, such as adding, replacing, transforming and removing parts. Also *view synthesis* – obtaining an image of a scene, given a set of images from other views of that scene – can be thought of as a form of image editing, in which we do not change the scene content, but the viewpoint. All of the above tasks have in common that they boil down to hallucinating the missing/occluded parts, in line with the visible scene context.In addition to the aforementioned generative challenge,

the manipulation task adds up new fronts. First, one needs to ensure that the change does not affect regions or aspects that we wish to preserve. In case of object transformation, it is often desired to preserve the identity of a certain object while changing its pose. In a view synthesis task, the goal is to keep the 3D scene consistent among different views. Second, manipulations of content or viewpoint frequently lead to revelation of originally occluded content. Exemplary, considering an image with a chair in front of a table – if we remove the chair from the room, or observe the room from a different angle, a region of the table that was not originally visible will be exposed, *i.e. dis-occluded*. This raises the question – what is the most effective way or representation for handling these scenarios?

To reason, for example, about occluded parts from a single image is an ambiguous problem, which cannot be solved analytically. Nonetheless, as humans we are still able to conduct such reasoning and even hallucinate occluded parts of objects [1, 132], based on lifelong experience of perceiving and interacting with similar objects. Similarly, deep learning has recently shown to be particularly strong at learning mappings for such ill-posed problems, when having access to an appropriate and large enough dataset. Exemplary, deep learning has achieved considerable success in generative tasks, thanks to advancements in generative models such as variational Auto-Encoders (VAE) [71] and Generative Adversarial Networks (GAN) [38]. With this motivation, in this dissertation we rely on deep learning to extract meaningful visual features from data in order to tackle the task of conditional scene synthesis and editing. Furthermore, based on the intuition that awareness of the object identities aids our brains at successfully completing invisible regions, prior to generating scene content, we attempt to explain the scene, *i.e. scene understanding*. This consists in perceiving and interpreting a scene in terms of semantics and geometry, to identify the underlying components, their location and interactions. Thereby, we explore different *compositional representations* of the scene, which enrich the knowledge about the underlying objects, while providing modular control (Figure 1.1).

First, we harness a representation that decomposes the scene into a set of RGB-D layers (Figure 1.1, b), such that multiple RGB-D values are available at each pixel location. The proceeding layers in each pixel can become exposed once the vantage point of the camera is changed and is, thus, suitable for generating novel views of the scene. Consider a user wearing a Head-Mounted Display (HMD), in which the perceived view responds to the user's head motion. In this scenario, the layered representation offers a good compromise between a single depth layer (2.5D) and 3D, as the former would lead to information holes for the dis-occluded surfaces (Figure 1.2), while a full 3D scene exceeds the needs of the HMD setting, *i.e.* most surfaces would never be visible from the slightly perturbed viewpoints. Our layered representation is based on the Layered Depth Image (LDI) proposed by [125] for efficient image-based rendering, which we will later discuss in detail in Section 2.1.2. Different to LDIs, we additionally exploit semantic information for layer forming, which in turn, as we will show later, enables the further application of object or foreground removal. In contrast to related works that rely on multiple views as input [45, 177], in this dissertation, we build a path towards monocular LDI prediction. The advantage is that such method does not rely on the availability of appropriate photo sets, so that the users can experience an immersive visualization out of any single photo. We initially introduce a simple two-layer

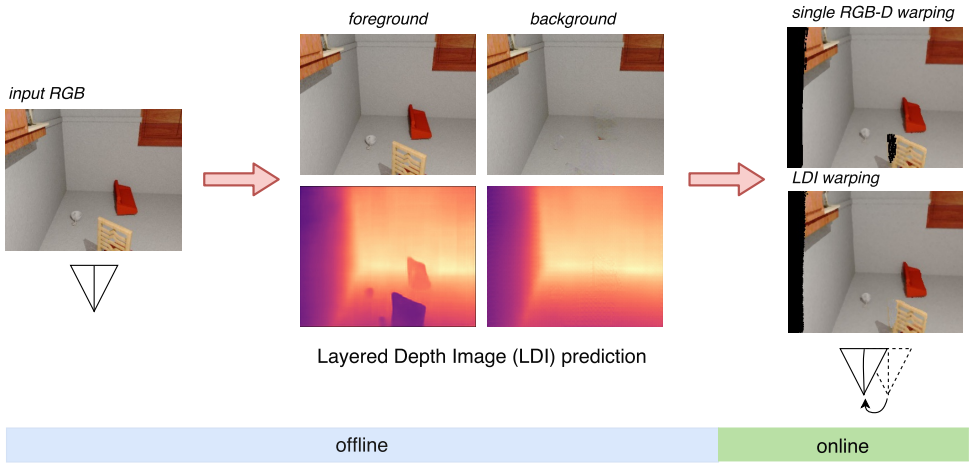


Figure 1.2. Viewpoint perturbation of an image comparing simple warping of a single RGB-D and rendering from a layered RGB-D representation (LDI). The LDI generation is done once and offline, prior to the online simulation of the viewpoint changes. [©2019 Pattern Recognition Letters]

approach [22] which decomposes the scene into a foreground and background part, and then inpaints the background to obtain a two-layer LDI. At the time of submission, this was the first work that investigated this task from monocular input. A concurrent work from Tulsiani *et al*. [139] tackled the same problem in a different formulation, where the LDI prediction is instead learned in a self-supervised fashion, through a view synthesis proxy task. We experimentally demonstrate in Section 3.4 that our approach delivers superior results compared to [139]. Second, motivated by the advantages of the proposed two-layer model over the view synthesis supervision alternative, we follow a similar decomposition idea as in [22], however utilize a more object-level understanding, which generates a layer per instance and completes the individual semantic layers of the scene [21]. This was the first work that generated a flexible number of layers depending on scene complexity, which further improved the generation performance for the occluded regions. Note that an alternative to LDI generation would be to directly perform view synthesis in each view. This however is less efficient as it invokes an inference step at every view change. In contrast, an LDI is generated once (offline), and the computation costs on each view change amounts to solely depth-based warping.

Second, we explore semantic decomposition of scenes via scene graphs (Figure 1.1, c), which are representations composed of objects labels as nodes and inter-object relationship labels as directed edges. Scene graphs have recently shown to be an appropriate interface for image generation [65] due to two desirable properties. First, they are abstract enough to allow for low-effort user interaction. Second, their interface is modular, which enables control on specific parts of the scene. Compared to dense semantic maps [146], scene graphs avoid a considerable amount of manual work from the user, *i.e*. tedious segmentation of each pixel. Additionally, they make it possible to specify the semantic relationship between two entities, which is not captured directly in a semantic map. Thereby, in this dissertation we want to explore the usage of scene graphs for the task of semantic image manipulation [19], in which the user can induce changes in an image by simply interacting with the nodes and edges of a graph. Thereby we face the challenge of partial image generation, combined

with preservation of the rest of the image. For this purpose we extend the vanilla scene graph representation – composed of semantic labels – to include visual neural features and bounding box coordinates for each object node. The extended representation allows to control what aspects of an instance need to be kept or altered during a transformation. For instance, changing a relationship label `girl - sitting on - horse` to `girl - beside - horse` requires that the appearance of the girl should be preserved, while the pose and location with respect to the horse change. Hence, here we would keep the visual features unchanged, and discard the box coordinates to allow the system to synthesize a valid new box that respects the new relationship constraint.

Inspired by the scene graphs' capabilities at generating and modifying images, our next research goal is to explore a similar semantic-assisted scene generation task for complete 3D scenes. In contrast to the image domain, in 3D we do not face the problem of projective ambiguity, which could facilitate the understanding of positional relationships. As suitable datasets for the task were not available at that time, we first created an appropriate large-scale dataset of 3D scenes with respective 3D semantic scene graphs. We obtain this dataset, named 3DSSG [141], in a semi-automatic fashion, extending from the real 3D environments of 3RScan [140]. As a proof of concept to depict the learning capabilities with respect to the collected dataset, we propose the first learned method that predicts a scene graph from the point cloud of a scene. Finally, we propose a system for simultaneous 3D scene generation and manipulation on the basis of these scene graph annotations [20]. Notice that a few works have very recently been proposed to generate a 3D layout (see Note below) from a scene graph [96], which is then populated by means of shape retrieval. Nevertheless, we are the first to explore a model for *fully-learned*, end-to-end scene generation. In particular, we demonstrate that shape and layout estimation are inherently correlated tasks, enhancing the performance of both components when learned simultaneously. Thereby, similar to the composition proposed in the image editing task, we extend the semantic scene graphs with 3D bounding boxes and neural features obtained from object shapes.

Note

Throughout this dissertation we will use the term *layout* with two different definitions, to adapt to the respective context in literature. In the context of layered depth images, layout refers to the empty room of a scene, *i.e.* the set of structural elements of the scene such as walls, floor, ceiling, etc. The reader should assume this definition in Chapter 3. In the context of the scene graph works, layout refers to a structure (skeleton) that describes the arrangement of parts in a scene. In 3D for instance, this boils down to the set of 3D bounding boxes of all objects in the scene. This definition should be assumed in Chapter 4.

1.2 Contributions

This section summarizes the contributions of this dissertation. The first two publications tackle the problem of monocular LDI generation:

[22] **H. Dhamo**, K. Tateno, I. Laina, N. Navab, and F. Tombari. "Peeking Behind Objects: Layered Depth Prediction from a Single Image". *Journal Pattern Recognition Letters (PRL).* 2019

[21] **H. Dhamo**, N. Navab, F. Tombari. "Object-Driven Multi-Layer Scene Decomposition from a Single Image". In: *IEEE International Conference on Computer Vision (ICCV).* 2019

The main contribution in [22] is the introduction of this novel task (concurrently with [139]) and the demonstration of state-of-the-art results, by means of a new method based on depth prediction and image inpainting. Further, in [21] we propose a solution to the design limitations of previous works – we decompose a given input image in a flexible number of LDI layers. We do so via a semantic-aware approach that considers the detected objects in the scene. As a technical contribution, we demonstrate that re-composing all parts back, for a comparison against the visible surface improves the performance.

In [19] we introduce the novel task of image manipulation via a scene graph, which presents a low-effort user interface to induce semantic changes in an image. The core technical contribution lies in a training strategy that circumvents that need for image pairs with changes for training:

[19] **H. Dhamo***, A. Farshad*, I. Laina, N. Navab, G. D. Hager, F. Tombari, C. Rupprecht. "Semantic Image Manipulation using Scene Graphs". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2020

The next two works pave the way for semantic scene graphs for 3D scene generation and editing:

[141] J. Wald*, **H. Dhamo***, N. Navab, and F. Tombari. Learning 3D Semantic Scene Graphs from 3D Indoor Reconstructions". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2020

[20] **H. Dhamo***, F. Manhardt*, N. Navab, F. Tombari. "Graph-to-3D: End-to-end Generation and Manipulation of 3D Scenes Using Scene Graphs". In: *IEEE International Conference on Computer Vision (ICCV).* 2021

First, we tackle the lack of appropriate data with scene graphs annotations by obtaining a corresponding dataset (3DSSG) in a semi-automatic manner, thereby having a high focus on rich and dense semantic annotations [141]. Furthermore, we propose the first learned method for estimating the 3D scene graph from a point cloud of a scene. In [20] we propose the very first work that is able to generate a 3D scene from a given scene graph in a fully end-to-end fashion, and demonstrate that joint learning of object shapes and 3D layout leads to improved results.

As a practical contribution, in many of the aforementioned works, we generated large-scale datasets [21, 22, 141], or dataset extensions [20] suitable for the evaluation of the proposed novel tasks, which were not available at the time of publication of the respective works.

Finally, we approach the task of unconditional scene generation via scene graphs:

[33] S. Garg*, **H. Dhamo***, A. Farshad, S. Musatian, N. Navab, F. Tombari. "Unconditional Scene Graph Generation". In: *IEEE International Conference on Computer Vision (ICCV)*. 2021

This work will not be discussed thoroughly in this dissertation, however, we discuss a system overview and the relevant implications for scene generation.

1.3 Structure of this Dissertation

This section briefly outlines the structure of this dissertation.

Chapter 2 sets the theoretical background for this work, including fundamental concepts in computer vision and deep learning. In essence, we describe the pinhole camera model which is assumed in our works, the LDI representation, as well as deep generative models.

Further, Chapter 3 presents the monocular LDI generation approaches investigated in this work. We first give an overview of the related work, then describe our methods for two-layer and object-driven LDI generation, and evaluate the methods by showing the usefulness in content editing and view synthesis.

Chapter 4 presents our work on scene graphs for scene synthesis and editing. We first discuss the related work, then introduce the methods on semantic image manipulation, generation of 3D scene graphs, and 3D scene synthesis/manipulation, together with the respective evaluations.

Chapter 5 briefly introduces an additional work carried out during this thesis, which offers interesting insights for future works in unconditional scene generation.

In the Appendix we provide additional findings and results, such as the application of the obtained 3D scene graphs in domain-agnostic scene retrieval A, supplementary results on the evaluation of the proposed object-driven layered scene decomposition B, more details and statistics on the 3DSSG dataset C, as well as further details and metric definition for 3D scene generation D.

1.4 Applications

This section aims to show the practical value of the work carried out during this thesis. We identify the following set of potential applications.

Immersive visual experience View synthesis is the direct application of the LDI representations generated in our works [21, 22]. When perturbing the original viewpoint, the additional layers support novel regions that were originally not visible. For instance, in Mixed Reality (XR) – assuming a user has access to reality only through a mounted camera in the HMD – it is of interest to warp the image obtained by the camera to the viewpoint of the user, so that the

scene is naturally perceived by the eyes. Moreover, one can obtain a 3D photo [45], in which the image responds to the slight viewpoint changes of the user head/device – technically a view synthesis step for each view perturbation – which gives a 3D effect to the perception of a standard photo.

Image editing software Editing photos is one of the most interesting tasks for visual artists, *e.g.* photographers, using commercial designer tools such as Photoshop. There is particular focus on smarter inpainting tools, such as content-aware fill. One step further for such a tool would be to incorporate semantic information in a user friendly manner, similar to our proposed graph-based semantic image manipulation work [19]. Alternatively, one could decompose the underlying image in layers [21] to offer a comfortable interface for object-aware removal.

Robotics The field of robotics can also benefit from our scene manipulation works [19, 20]. For instance, a robot instructed to tidy up a room can first manipulate the scene graph of the perceived scene, *e.g.* changing their relationships and attributes: "clothes lying on the floor" to "folded clothes on a shelf", to obtain a realistic future view of the room.

Scene retrieval In [141] we show that the generated scene graphs can be used for domain-agnostic scene retrieval, where the graphs of a 3D scene or an image are used as query to retrieve the most similar 3D scene from a database. This can be useful for instance, for localizing a robotic agent or human, or providing the user with a room that best matches the expectations. More details on this retrieval task can be found in appendix A.

Theory and Fundamentals

2

2.1 Computer Vision

2.1.1 The pinhole camera model

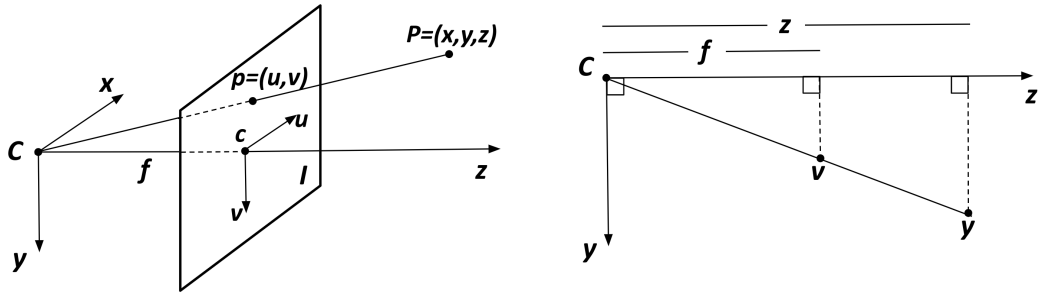


Figure 2.1. **The Pinhole Camera Model** describes the perspective projection of the 3D points onto the image plane. *I:* image plane, *C:* camera center, *c:* optical center, *f:* focal length, *P:* point in 3D space, *p:* 2D projected point.

All works presented in this dissertation follow the *pinhole camera model* to relate 3D points and the 2D image plane. Given the world and a camera, located at position C, the pinhole camera model describes a mapping between a 3D point $P = (x, y, z)^T$ in the world and its respective projected point $p = (u, v)^T$ on the image plane I of the camera [41]. Let the focal length f be the distance from the camera center C to the image plane I. Following the similar triangles rule, (*c.f.* Figure 2.1 (right)), the following equations can be derived $\frac{v}{f} = \frac{y}{z}$, $\frac{u}{f} = \frac{x}{z}$ from which we can infer $v = f\frac{y}{z}$ and $u = f\frac{x}{z}$. The image plane origin is usually defined at the top-left corner of the image. Therefore, the points after projection are shifted by the optical center $c = (c_x, c_y)$, leading to $v = f\frac{y}{z} + c_y$ and $u = f\frac{x}{z} + c_x$.

Commonly, *homogeneous coordinates* are utilized to turn the perspective projection into a linear system, and therefore simplify the notation and many related derivations. This is done by adding a fourth coordinate to a 3D point $p = (x, y, z, 1)^T$. In homogeneous representation all points $P = (\lambda x, \lambda y, \lambda z, \lambda)^T$ with $\lambda \neq 0$ represent the same euclidean 3D point. For $\lambda = 0$, this formulation describes a 3D point which lies at infinity. These points live in the projective space $\mathbb{P}^3$. The associated 3D point can be obtained through a simple division by the last coordinate $(x, y, z, k)^T \widehat{=} (\frac{x}{k}, \frac{y}{k}, \frac{z}{k})^T$. Notice that unlike euclidean spaces, projective spaces can

explain points at infinity ($k = 0$). The perspective projection can thus be written as a linear transformation

$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f\frac{x}{z} + c_x \\ f\frac{y}{z} + c_y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = KP. \tag{2.1}$$

Here K denotes the camera *intrinsics* matrix, which contains the parameters of the perspective projection. Equation 2.1 describes a camera that is located at the center of the world coordinate system, which is, most commonly, not the case. Therefore, one needs another matrix T to describe the transformation from world space to camera space. T is known as the *extrinsics* matrix, or camera pose, and contains its rotation and translation in 3D world space $p = KTP$. In projective space, T is a 4×4 matrix that can be written as

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{2.2}$$

where R is a 3×3 rotation matrix, while t is a 3 dimensional translation vector.

Given two cameras and their respective intrinsics K_1, K_2 and extrinsics (camera poses) T_1, T_2, utilizing the pinhole camera model, one can warp the image obtained in camera 1, to the viewpoint of camera 2. To perform this warping, we need to have per-pixel distance from the image plane of the source camera, to the 3D points, *i.e.* depth d_1

$$p_2 = K_2 T_2 T_1^{-1} (K_1^{-1} p_1) \cdot d_1 \tag{2.3}$$

where p_2 is further divided by its last coordinate to obtain the final 2D point. Often pixel-wise depth information is stored in an organized 2D structure with the same resolution as the respective image, also known as a *depth map*.

2.1.2 Layered depth images

Layered Depth Images (LDI) [125] encompass a scene representation that contains multiple depth values along each ray line in a single view. It was introduced as an efficient image-based rendering method, emerged in times of limited computational power. When rendering in a perturbed view, some scene structures that are occluded in the original camera frame get exposed – also known as dis-occlusion – and therefore populate the target view, Figure 2.2. The number of layers is different in every pixel and depends on the scene complexity. For small view perturbations from a vintage point, rendering from a 3D scene mesh is considerably more expensive and contains more data than needed. Technically, the LDI data structure is represented as a 2D array of layered depth linked lists of varying size. Each element in the list contains RGBA information, depth and a splat index.

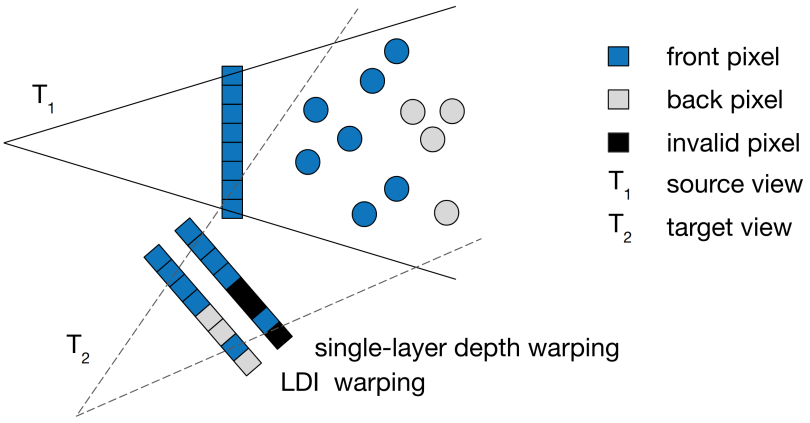


Figure 2.2. Layered depth image in source (reference) view T_1. When the view is perturbed T_2, the empty (black) pixels are filled with information from the consecutive layers (gray), illustrating the advantage over a single layer depth.

Traditionally, LDIs can be obtained by warping a set of images into a common view, or alternatively, by using a ray tracer to sample points from a scene view in every line of sight [125]. The latter offers the advantage of controllable sampling density, while the former is constrained to the data availability in the given image frames. Nevertheless, in real world scenarios, where we do not know the 3D scene and are limited to a set of multi-view images, only the former alternative is applicable.

To render an LDI in a different view-point, [125] leverage an efficient incremental warping approach, following the pinhole camera model. Let T_1 be the camera matrix for the LDI, which projects a 3D point from the world coordinates system into the camera's pixel coordinate system. Similarly, we define T_2 as the target view camera matrix. Then, the transformation matrix between the two views is $T_{1,2} = T_2 T_1^{-1}$, which maps a point $p_1 = (x_1, y_1, z_1, 1)$ to $p_2 = (x_2, y_2, z_2, 1)$ using homogeneous coordinates

$$p_2 = T_{1,2} p_1. \tag{2.4}$$

Exploiting the linearity of this operation, one can rewrite it as

$$T_{1,2} p_1 = T_{1,2} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = T_{1,2} \begin{bmatrix} x_1 \\ y_1 \\ 0 \\ 1 \end{bmatrix} + z_1 \cdot T_{1,2} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \text{start} + z_1 \cdot \text{depth}. \tag{2.5}$$

Moving along a scanline, *i.e.* pixels with the same y_1 and increasing x_1, the start component at pixel $(x_1 + 1, y_1)$ can be computed from the start component at pixel (x_1, y_1) as

$$\text{start} = T_{1,2} \begin{bmatrix} x_1 + 1 \\ y_1 \\ 0 \\ 1 \end{bmatrix} = T_{1,2} \begin{bmatrix} x_1 \\ y_1 \\ 0 \\ 1 \end{bmatrix} + T_{1,2} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \text{start} + \text{xincr.} \tag{2.6}$$

This decomposition and incremental computation of the matrix operations reduces unnecessary computational costs, shared among different pixel locations.

For each layered depth pixel, the computation of the target pixel coordinates is done in back-to-front order. Evidently, the color values are warped from the source to the target pixels. Here the splat size is computed as a function of the distance from the surface point to the T_1 camera, as well as the angle between the surface normal and the line of sight.

Noteworthy, very related to LDIs are Multiplane Images (MPI) proposed by Zhou *et al*. [177], which leverage multiple RGBA layers of fixed depth. MPI can be thought of as a discrete version of the LDI, where the number of layers need to be pre-defined explicitly, in contrast to the LDI.

2.2 Deep Learning

At the heart of each method proposed in this thesis lies deep learning. This section starts with a historical overview to place modern deep learning in the right context. Further, it introduces the fundamentals of deep neural networks, such as their composition and optimization. Finally, the section concludes with an overview on convolutional neural networks (CNN) – which we heavily rely on in this work – and their most relevant building blocks.

2.2.1 Brief historical overview on deep learning

Despite the very recent outburst, deep learning is not a new concept. There have been three waves of deep learning, starting from 1940. Below, we give an overview of each of these waves, referring the reader to [37] for more details.

The first wave is known as *cybernetics* (1940 - 1960), essentially referring to linear models motivated by neuroscience. McCulloch and Pitts [99] introduced the artificial neuron, which could recognize two different classes of inputs, resulting in a positive or negative answer. Note that here the weights were not learned. Instead, they had to be set by a human operator. The perceptron from Rosenblatt [119], is the first model that could learn a single neuron, given pairs of data examples from each category. However, soon after many limitations of the

linear models were discovered, such as, for instance, the inability to learn the XOR function. Such limitations significantly stalled the progress in AI, leading to the first winter of AI.

The second wave (*connectionism*), emerged in the 1980s. The core idea is that simple computational units can learn more complex concepts when combined together in a network. In its core lied the idea of distributed representation [49], which aimed to employ neurons that learn shared concepts, *e.g.* color, among different categories, to reduce the number of necessary computational units. Interestingly, the *back-propagation* algorithm was introduced during this time, which is today still an integral part of the training of almost all deep networks [120]. Regardless of this advancement, the second winter of AI emerged in the 1990s, as the initial expectations were unable to be accomplished in practice, leaving the AI investors disappointed.

Last, the still ongoing third wave of deep learning began with the introduction of deep belief networks from Hinton *et al.* [48] in 2006. They proposed to employ greedy layer-wise pre-training, an efficient training strategy that was later successfully incorporated in other models. Importantly, due to these findings, combined with increasing computational power, researchers were able to train deeper networks and started to understand the theoretical importance of network depth. The rise of deep networks led to a remarkable boost for many tasks in computer vision. Notably, in year 2012 AlexNet [74] – a deep convolutional network trained on two GPUs – significantly outperformed other methods on the ImageNet [18] classification challenge.

2.2.2 Neural network fundamentals

Neural networks consist of multiple layers of computation nodes, also known as *neurons*, inspired by the biological brain. Thereby, the neurons of the first layer represent the *input layer*, while the neurons of the last layer form the *output layer*. The remaining nodes, corresponding to the intermediate layers are called *hidden layers*. The nodes of adjacent layers most commonly have connections between them. However, there are cases where non-neighboring neurons are also connected [43]. A single layer in the network – referred to as *perceptron* – aggregates the information from the input units by multiplying the inbound signal with *weights*, and adding a free term, referred to as *bias*. The weights and bias values are learned during the training process.

For a given neuron z_j in layer l we write

$$z_j = \sum_i w_{ji}^{(l)} x_i + b_j^{(l)} = w_j^{(l)T} x + b_j^{(l)} \tag{2.7}$$

whereas for the whole layer we have

$$z = W_l x + b_l. \tag{2.8}$$

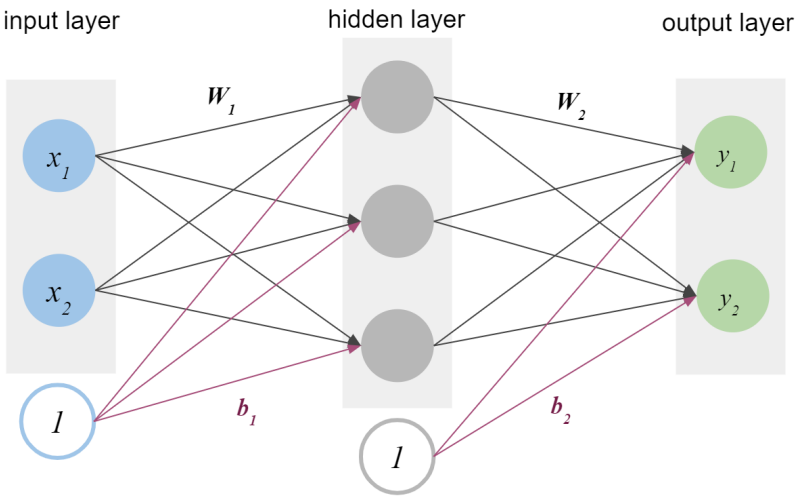


Figure 2.3. A simple neural network architecture with two layers, *i.e.* one hidden layer

Here, x_i is the result of a preceding layer that is connected with z_j, w_{ji} are the layer weights and b_j constitutes the bias term. $W_l \in \mathbb{R}^{d_{in} \times d_{out}}$ and $b_l \in \mathbb{R}^{d_{out}}$ are the weights and bias in matrix form, where d_{in} and d_{out} are namely the size of the input and output vector ($x \in \mathbb{R}^{d_{in}}$, $z \in \mathbb{R}^{d_{out}}$). Similarly, from the relation in Eq. (2.7) one can simply deduce the formulation for a model with two layers, as depicted in Figure 2.3

$$y(\mathbf{x}, \theta) = W_2 h(W_1 x + b_1) + b_2. \tag{2.9}$$

where $\theta = \{W_1, W_2, b_1, b_2\}$ describe the parameters of the network. Here h denotes the *activation function*. The purpose of an activation is to introduce non-linearity in the neural network. Notice that, if we omit h from Eq. (2.9), one can rewrite it as

$$y(\mathbf{x}, \theta) = W_2(W_1 x + b_1) + b_2 = W^* x + b^*, \tag{2.10}$$

which is equivalent to a single-layer network with weights $W^* = W_2 W_1$ and bias $b^* = W_2 b_1 + b_2$. Common choices for the activation layer include the sigmoid function, the Rectifier Linear Unit (ReLU) and its variants [15, 44], as well as the hyperbolic tangent (tanh).

Optimization

Having defined the structure of the neural network, the objective is to optimize all the learnable parameters θ (*i.e.* weights and biases). In a supervised learning context, given pairs of input and output samples (x_i, t_i), *i.e.* training dataset, this is done by feeding the input values x_i to the neural network and finding the parameters θ that minimize an error function between the predicted value and the *known* ground truth target t_i. This process is referred to as the *training* of the network. Thereby, an *epoch* represents a single pass through the entire training dataset. For a regression problem a common loss function is the $\mathcal{L}_2$ norm

$$\mathcal{E}(\theta) = \sum_i \|y(x_i, \theta) - t_i\|_2 . \tag{2.11}$$

The goal of minimizing the error function, also known as cost or loss, is reached through an iterative optimization process, which aims to find the minima of the error function, *i.e.* locations in $\mathcal{E}(\theta)$ where the error gradient is 0, $\nabla\mathcal{E}(\theta) = 0$. Certainly there can be more than one minimum within the error function, and this is usually the case, given that the model is non-linear, *i.e.* the error function non-convex. The lowest possible value in $\mathcal{E}(\theta)$ is typically also referred to as its *global minimum*. In practice, it is fairly impossible to converge to the global minimum, but a good local minimum suffices in most applications.

There are various methods used to approach the zeros in the gradient of the error functions, including steepest descent and conjugate gradient. The core idea is to go in the direction of $-\nabla\mathcal{E}(\theta)$ in a particular time step τ. The corresponding weights update is given by

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \eta \frac{\partial \mathcal{E}}{\partial \theta^{(\tau)}} \tag{2.12}$$

where η denotes the learning rate. The learning rate η is commonly a very important hyper-parameter of the optimization. If the learning rate is too small, it can lead to very slow convergence, while a too large learning rate could cause continuous oscillation around an optimum without ever reaching it.

The data samples can be fetched all at once, in mini-batches, or one at a time. The bigger the batch, the more robust the weight updates are with respect to noise in individual input samples. However, in deep networks memory bounds set a limit in the batch size that can be used to train. Therefore, a mini-batch variant of gradient descent is leveraged for the optimization, which is called *stochastic gradient descent (SGD)*. More recent optimization techniques exist, which rely on adaptive mechanisms and are more commonly employed in practice. For instance, SGD with momentum [113] computes the gradients of a step as a combination of the previous and the current gradients. This accelerates training when gradients of consecutive steps point the same direction, and slows it down when the directions change, *i.e.* to avoid oscillations around a minimum. Other optimizers, such as RMSprop, Adadelta [164] and Adam [70] incorporate adaptive learning rates for each parameter, relying on a history of past gradients. They reduce the need for hyper-parameter tuning of the learning rate and are particularly successful for noisy data.

Back-propagation

Now we dive into the problem of gradient estimation in a deep network architecture. This is typically done using the so called *back-propagation* algorithm, which is an efficient way for gradient computation. To understand why this is the case, we formulate the solution for the error minimization. In the two-layer network example, we are interested in minimizing

$$\mathcal{E}(\theta, x, t) = \sum_{i=1}^{N} ||W_2 h(W_1 x_i + b_1) + b_2 - \mathbf{t}_i||_2. \tag{2.13}$$

Relying on a gradient based method for the weight update, we are required to compute the gradients of the cost function with respect to θ

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{\partial}{\partial \theta} \sum_{i=1}^{N} ||W_2 h(W_1 x_i + b_1) + b_2 - \mathbf{t}_i||_2. \tag{2.14}$$

We observe that the forward pass has the structure of a function composition of the form $l_2(l_1(x))$, with $l_1(x) = h(W_1 x + b_1)$ and $l_2(x) = (W_2 x + b_2)$ being each layer of the neural network. Therefore, we can apply the chain rule for gradient computation of the cost function with respect to the parameters and obtain

$$\frac{\partial \mathcal{E}}{\partial W_2} = \frac{\partial \mathcal{E}}{\partial l_2} \frac{\partial l_2}{\partial W_2}, \qquad \frac{\partial \mathcal{E}}{\partial W_1} = \frac{\partial \mathcal{E}}{\partial l_2} \frac{\partial l_2}{\partial l_1} \frac{\partial l_1}{\partial W_1}. \tag{2.15}$$

Notice that, some terms of the derivative computation with respect to W_1, are already computed for W_2. A similar observation can be deduced for the bias terms. This means that the error value in a particular hidden neuron is given by propagating back the errors of the proceeding adjacent layers with whom this unit shares a connection. The shared reusable terms can be stored, to avoid re-computation and hence make gradient computation more efficient.

We illustrate the presented training procedure in a nutshell:

Training epoch of a neural network

1. Forward propagate data samples $\mathbf{x}_i$ in the neural network, using the current parameters θ
2. Compute the difference between the ground truth $\mathbf{t}_i$ and the predicted output $\mathbf{y}_i$
3. Compute the derivatives $\frac{\partial \mathcal{E}_i}{\partial \theta}$ for each weight from the propagated errors
4. Update the weights in the direction of steepest decent of the error gradient

Having obtained a set of parameters that minimize the cost function from training, the neural network is able to predict outputs for new input samples that it has never seen before. This process is also known as *testing* or *inference*.

2.2.3 Convolutional neural networks (CNN)

For certain computer vision applications, based on a structured input, *e.g.* image, one wants to enforce particular properties in the learned model that are hard to capture with standard neural networks (MLPs). For instance, recognizing certain shapes or features in images should be based on a *local context* and not in the entire image. Moreover, the extraction of relevant features should be independent on the exact location in the image. For example, consider the problem of image classification. A `cat` should still be classified as a `cat`, regardless of its location in the image. Since linear layers consider pairings between each

input and output node, it can become quite costly when dealing with high dimensional input data such as images. With this motivation arises the need for weights that capture only local structures around a pixel. Further, the location independence allows for *weight sharing* across spatial locations, which significantly reduces the number of parameters. This is the basis for the *convolutional neural networks* [78]. Here, the weights are commonly referred to as *filters* or *kernels*, due to the resemblance with a correlation problem, where one of the images is a small window sliding along the other one, resulting in a map of dot products. Like the usual neural network, each convolutional layer proceeds with an activation function to introduce non-linearity.

Building blocks of a convolutional neural network

Fully-connected Layer. The most basic block is a fully-connected layer, also known as a linear layer. Same as in a regular neural network, every input neuron is connected with all the output neurons. As previously discussed, a fully connected layer computes the weighted sum of all input parameters and adds a bias to the output according to $y(x) = Wx + b$ with $W \in \mathbb{R}^{d_{in} \times d_{out}}$ and $b \in \mathbb{R}^{d_{out}}$.

Convolutional Layer. The convolutional layer can be considered the most identifying layer of a CNN, as it enables its desired weight sharing and local connectivity properties. In a 2-dimensional scenario, a convolution is carried out through a set of d_{out} filters with $W_i \in \mathbb{R}^{d_{in} \times k_x \times k_y}$ and $i \in \{1, ..., d_{out}\}$, *i.e.* the learnable parameters of the layer. Thereby, d_{in} denotes the depth of the input feature and k_x, k_y represent the spatial filter size. Each filter is spatially slid across the input feature of size $M \times N \times d_{in}$, computing a dot product between the filter values and the corresponding local region in the input feature, resulting in an output feature of size $M \times N$. The final output volume $M \times N \times d_{out}$ for this layer can be obtained by stacking all d_{out} output feature maps. Another relevant parameter for a convolution is the *stride*. A stride of s can reduce the output resolution by a factor of s, by only applying the convolution at every s-th location in the input feature. One can optionally add a bias to the convolved feature maps. Similarly, it is possible to employ a 1-dimensional or 3-dimensional convolution, by adjusting the number of dimensions in the filters.

Pooling Layer. Pooling layers [78] are intended for down-sampling (sub-sampling) the input feature map, similar to the stride parameter for convolutions. By reducing the spatial size of the feature maps, it controls the capacity of the network and introduces some invariance to small transformations of the input, as very precise local information is lost. Since the exact locations of features are not relevant in most tasks, it is often sufficient to only maintain the relative locations. Each feature map is convolved with a filter of size $k_x \times k_y$, which forwards the desired signal within the filter. This filter is not learnable and can be set explicitly as a *maximum*, *averaging* or even *L2-norm* operation. Similarly to a convolution layer, another hyper-parameter of the pooling layer is the stride. Notice that for max-pooling the gradient is simply set to flow through the input with the maximum value, since the $max(\cdot)$ operation is not differentiable.

Activation functions. Convolution layers are usually followed by non-linear activation functions. Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ and tanh $th(x) = 2\sigma(2x) - 1$ are activation functions

that limit the output range to namely $[0, 1]$ and $[-1, 1]$, which is convenient for mapping continuous input values to a more interpretable value, for example, closer to a *yes* or *no* answer. However, these functions encounter a saturation problem, as their gradients become very small as we diverge from the middle range. Therefore, more practically Rectified Linear Unit (ReLU) $f(x) = \max(0, x)$ are used for intermediate (hidden) layers, while sigmoid and tanh are often used for output layers to constrain the range of output values. Notably, ReLU does not saturate which, thus, speeds up convergence. Nevertheless, ReLU can drive many neurons in the inactive region (zero output and gradients). This led to alternatives such as leaky ReLU [44] or exponential linear unit (ELU) [15] which have a softer effect on the negative neurons.

Feature Normalization. In deep neural networks feature normalization is a necessity, due to the common *vanishing gradient* problem. The multiplication of mostly small terms in the partial gradient computation leads to very small updates for early layers. Normalization thus keeps the features of each layer at a stable range, avoiding such gradient decay.

Many normalization techniques have been proposed [6, 56, 74, 152] in literature, such as *Local Response Normalization* (LRN), introduced with AlexNet [74]. Batch Normalization (BatchNorm) is, however, the most popular approach [56], and also extensively employed in the works presented in this dissertation. BatchNorm computes moments across the mini-batch dimension, which are used to update an exponential moving average and standard deviation. To accomplish this, [56] first uses mini-batch statistics to map the features to a mean of zero and standard deviation of one. For every dimension k of an input x we have

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}. \tag{2.16}$$

In some cases, this whitening of the features, *i.e.* mean of zero and unit standard deviation, is not appropriate to represent the layer. Considering the output of a ReLU activation, which dampens the negative values, zero is not a good choice for the mean value. Therefore, a learnable scaling and offset are applied on the normalized value:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}. \tag{2.17}$$

During inference, it is desired that the output is entirely determined by the input. Hence, we normalize using the population mean and variance instead of relying on mini-batch statistics. Noteworthy, Wu and He [152] have shown that BatchNorm does not work well for small batch sizes. Therefore, they propose GroupNorm, computing the statistics through the channels dimension in groups of a given size.

Common CNN architectures

This section introduces some common CNN architectures, which are also used in this work.

Residual networks. He *et al.* [43] introduced ResNet, which is based on residual connection blocks, with the motivation to address the vanishing gradient problem [43]. We denote with H the mapping that a neural network block learns in a standard form $H = f(x)$. Instead, a

ResNet block learns the residual mapping $H = x + f(x)$, by applying an addition operation between the input of the block and the result feature of the block. Such residual connections come with two main benefits. First, learning a small residual x is a much easier task than directly learning the whole mapping H. Second, this optimizes the gradient flow as gradients can take a shortcut through the residual connection around it, allowing to train very deep networks. In this work, we employ a ResNet-50 architecture as a backbone for various tasks, such as RGBA-D object completion [21] as well as joint depth and mask prediction [22].

Skip connections. The U-Net architecture [118] was first proposed for the purpose of precise image segmentation. Since then, it has been widely adopted for a diverse set of image translation tasks [58]. In its core lie the *skip connections*, which consist in concatenating features of the same spatial size from the encoder and decoder, prior to being processed by the proceeding decoder layer. This enables a combination of high and low level features which has proven to improve the sharpness of the outputs. In this dissertation, we employ a U-Net-like architecture for RGB-D background inpainting [21, 22].

2.3 Generative Models

In terms of what they model, neural networks can be classified in two main categories, *generative* and *discriminative* models. A discriminative model learns the conditional probability of a target given the input data. For instance, provided an input image, a discriminative model can be trained to predict the category of the object in the image, or regress the depth values for each pixel, provided the respective target ground truth data. Instead, a generative model, aims to learn the probability of the data itself, in the form of common patterns and regularities in the data and the likeliness of certain samples to (co-)occur. Once trained, generative models can be used to synthesize new data, which fairly represent the real data distribution. In this dissertation, we employ generative models and their conditional variants for a set of tasks, such as image inpainting [21, 22], scene generation [20] and manipulation [19, 20], as well as unconditional scene graph generation [33]. The sections below present Variational Auto-Encoders and Generative Adversarial Networks, as they are also used in the work presented in this dissertation.

2.3.1 Variational auto-encoders

Auto-Encoders f_{ae} refer to a class of neural networks that aim to learn a compact and efficient representation of data, also known as latent code. Since no labels are required for training of an Auto-Encoder, they constitute a classic representative for *unsupervised learning*. Essentially, the latent codes are learned by means of a reconstruction loss (*e.g.* $\mathcal{L}_2$ norm), trying to reproduce the input data with $\mathcal{L}_r = ||f_{ae}(x) - x||_2$. The network architecture consists of two parts. The *encoder* receives the raw data and computes the latent codes. The *decoder* then receives the latent codes to reconstruct the original data. Typically, Auto-Encoders provide a good approximation of the original data, however, they cannot perform a one-to-one copy of

the input, as the latent codes only extract the relevant information from the data. Thus, they serve as data compression, and some denoising effect is expected.

It is very difficult to use a vanilla Auto-Encoder as a generative model. Since no assumptions can be made on the distribution of the learned latent codes, appropriate sampling cannot be conducted easily. This problem is addressed by a variational variant of Auto-Encoders (VAE) [71]. The core idea is to regularize the distribution of the latent codes, such that it is regular enough to sample. A VAE is similar in structure to a vanilla Auto-Encoder, as it is composed by an encoder and decoder network. Differently, instead of a single vector, they aim to learn a distribution on the latent vectors from the data, which facilitates later sampling. Practically, most often this distribution is assumed to be Gaussian, and the encoder network returns a mean μ and a covariance matrix σ to describe such distribution. After sampling a point from this distribution, one can decode it to the high-dimensional data. To do so, in addition to the reconstruction loss on the input (high-dimensional) data, VAEs incorporate a Kullback-Leibler divergence term on the latent code level, as a regularization between the learned Gaussian distribution and a standard Gaussian

$$\mathcal{L} = \mathcal{L}_r + D_{KL}(\mathcal{N}(\mu, \sigma), \mathcal{N}(0, 1)). \tag{2.18}$$

Note that at training time, one needs to back-propagate the error gradients all the way to the input nodes. However, the sampling operation from the learned distribution $z \sim \mathcal{N}(\mu, \sigma)$ is not differentiable. Therefore, VAEs are practically used with a common re-formulation, better known as *reparameterization trick*, making the network fully differentiable. The core idea is to transfer the sampling at an input node, by introducing a new parameter $\epsilon \sim \mathcal{N}(0, 1)$. The random variable z is then obtained as a function of the learned distribution and ϵ, according to $z = \mu + \sigma \odot \epsilon$, where $\odot$ denotes element-wise multiplication. In this form, the VAE is fully differentiable.

We refer the reader to [71] for a mathematical formulation of VAEs and derivations.

2.3.2 Generative adversarial networks

Generative Adversarial Networks (GANs) [38] refer to a generative model, which in essence is composed of a *generator* G and a *discriminator* D, trained with conflicting objectives. The goal of the generator G is to synthesize data that is not distinguishable from the real data. On the other end, the discriminator D is trained to distinguish if a given sample is synthesized (fake) or taken from the real data distribution (real)

$$\mathcal{L}_{GAN} = \min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]. \tag{2.19}$$

These two opposite objectives can push the generator to continuously improve its modeling of the true distribution, resulting in more and more realistic synthesized samples. GANs can be simply extended to conditional variants [8], which led to their popularity in many computer vision tasks, such as image inpainting [55, 108], image editing [180], style transfer

[58, 81, 181], super-resolution [79] and 3D object generation [151]. Isola *et al.* [58] propose a general–purpose conditional GAN that can be used for a variety of image translation tasks, such as labels to images, gray scale to color images, contours to image and incomplete to full image. The conditional GAN objective can be represented as

$$\mathcal{L}_{\text{GAN}} = \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_{x}[\log(1 - D(x, G(x)))] \tag{2.20}$$

where x represents the conditioning input and y the corresponding target image. In this formulation, G optimizes the following objective

$$\hat{G} = \min_{G} \max_{D} \mathcal{L}_{\text{GAN}} + \lambda_{\mathcal{L}_1} \mathcal{L}_1(y - G(x)). \tag{2.21}$$

which combines a GAN term and a L1 term with $\lambda_{\mathcal{L}_1}$ being their balancing factor.

Compared to VAEs, GANs commonly result in more realistic and crisp images, which makes them the most popular image-based generative model to date. On the other hand, VAEs are generally more stable to train and less data-hungry than GANs, while being more interpretable as they explicitly learn a distribution of latent codes.

Part II

Compositional Representations for Synthesis and Editing

Layered Depth Image Prediction

3

Our objective is to learn a mapping from a single RGB image to an LDI, *i.e.* a layered RGB-D representation of the scene. The task encompasses a set of open problems in computer vision, such as monocular depth prediction, and completion of occluded segments. In this dissertation, the LDI generation problem is formulated as a system that first understands the scene, *i.e.* identifies the set of visible entities in the image, and then completes each entity with the respective occluded regions. We start with a simple approach that decomposes the scene in a foreground and a background layer, which will be detailed in Section 3.2. Throughout this dissertation we will refer to this two-layered model as *PBO* (**P**eeking **B**ehind **O**bjects). Unsurprisingly, such two-layered model is often not sufficient to capture all the occluded structures in the scene. Exemplary, if at a certain image location there is a chair occluding a table, which is in turn occluding a wall, multiple levels of dis-occlusion (revelation) can eventually take place, which need more layers to be fairly represented. Therefore, building on a similar structure, we formulate an object-driven approach which decomposes the scene into a set of layers, representing the empty background (layout) as well as each identified object in the scene (Figure 3.1), described in Section 3.3. This method – which I will refer to as *OMLD* (**O**bject-driven **M**ulti-**L**ayer **D**ecomposition) – generates a flexible number of layers, depending on the complexity of each scene. Both methods are evaluated at the time of publication, against state-of-the-art baselines. Section 3.4 reports the results on LDI generation, novel view synthesis as well as diminished reality.

3.1 Related Work

3.1.1 Layered representations

Scene representations that decompose a scene in layers are explored in a variety of formulations, such as depth ordering of semantic maps [57, 136, 159] and amodal color images [23], optical flow [134, 142], stereo reconstruction [7], scene decomposition in depth surfaces [91] and planes [90].

The focus of this thesis is on the Layered Depth Images (LDI) by Shade *et al.* [125], which refer to an extended single-view representation of a scene that contains multiple RGB-D values per pixel. As discussed in Section 2.1.2 LDIs were first proposed for efficient image-based rendering on view perturbations, to deal with information holes on dis-occlusion. Applications include 3D photography [45] and video view interpolation [182]. Hedman *et al.* [45] use such a representation to reconstruct a 3D photo from multi-view inputs, captured

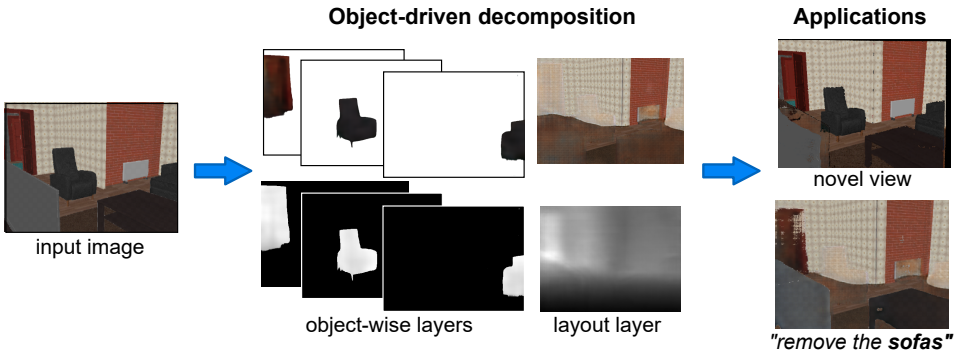


Figure 3.1. Starting with an image, OMLD decomposes the scene into a set of RGBA-D object layers and layout layer. The representation can be used for view synthesis and object removal. [©2019 IEEE]

from a set of hand–held camera images. The different views are stitched together in a panorama LDI of two layers. Zhou *et al*. [177] infer multi-plane images (MPI), a similar representation from stereo input, that decomposes an image into sweep planes with fixed depth. Concurrently with [139] we proposed the first work for LDI prediction from a single RGB image [22]. Thereby, Tulsiani *et al*. [139] propose an indirect supervision with a view synthesis proxy task. Though naturally under-constrained, monocular LDI generation has the advantage of enabling the 3D enhancement of any given photo, even if additional views or depth information is not available. In contrast to [139] we formulate the problem as a combination of scene understanding (depth prediction, background segmentation) and inpainting. Further, we exploit an object-driven approach [21], which in contrast to previous monocular LDI generation methods generates a flexible number of layers. Succeeding our works, LDI [128] and MPI [137] based approaches, relying on monocular input were proposed, for namely 3D photography and view synthesis.

3.1.2 View synthesis

The task of predicting the depth or 3D structure of a scene is directly related to novel view synthesis and we later demonstrate our LDI prediction within this challenging application. View synthesis consists in generating an image from an unseen view of a scene, utilizing image information from known viewpoints. A recent line of work on synthesizing new views directly minimizes a reconstruction loss between the reference and the target image in an end-to-end manner [31, 154, 178]. Depth prediction often arises as an implicit, intermediate representation in such frameworks when using view pairs as input or supervision [32, 36, 154, 176]. At the time of publication of [22], Zhou *et al*. [178] was the most comparable method, as it does not rely on multi-view inputs or additional geometry. There have been view synthesis approaches, more recent than our works, which exploit MPIs [137] or point clouds [149] as intermediate representations, and rely on differentiable/neural rendering to obtain the final image result.

3.1.3 Monocular depth prediction

One of the first approaches for depth estimation from a single image involved hand-engineered features and inference using Markov Random Fields (MRF) [123, 124]. Later, data-driven methods were proposed that retrieve and warp similar samples from a database [68, 72]. With the rise of deep learning, depth prediction from a single image became a popular and active research field. Here we mainly discuss works that predict depth in a supervised fashion. Eigen *et al*. [25] were first to propose a multi-scale CNN with a coarse and fine component, further extended to other modalities such as normal maps and semantic segmentation [24]. Further, Chakrabarti *et al*. [9] exploit CNNs to estimate depth as a combination of depth derivatives of different orders. Deeper fully-convolutional networks [69, 76] were then proposed, namely based on a ResNet [43] and DenseNet [53] architecture. Commonly, Conditional Random Fields (CRFs) [80, 92, 93, 145, 155] are used as a way to enforce geometrical constraints. Another line of works exploit semantics to facilitate depth prediction [61, 89]. Other works, more recent than our publications [114, 115] focus on improving the performance on the depth contours, as smooth edges is a commonly observed problem in CNN-based depth prediction.

Deep networks have additionally been leveraged to generate 3D information from a single image, as a single 3D object [14, 27, 150, 151], or a factored 3D scene [138]. Different from our approaches, these methods do not focus on texture generation in the occluded regions. Other related works exploit more extended inputs to predict 3D scene representations, such as a panorama image [183], or depth information [40, 131, 148].

3.2 Two-Layered Model (PBO)

In PBO we break down the LDI prediction task into two subsequent steps, which are presented in Figure 3.2. First, given an input RGB image I, we jointly learn a standard depth map d as well as a binary mask m_{BG} (Section 3.2.2) indicating the background pixels in the image. Then, the binary mask is multiplied with the RGB image and the predicted depth map, to extract the not occluded background regions $(\tilde{I}_{BG}, \tilde{d}_{BG})$, which are fed to the following step. Second, we employ RGB-D background completion via a deep GAN network, conditioned on the masked modalities $(\tilde{I}_{BG}, \tilde{d}_{BG})$, which yields the background layer (I_{BG}, d_{BG}) as explained in Section 3.2.3. The resulting LDI, comprised of two layers, is thus given by the quadruple (I, d, I_{BG}, d_{BG}). To evaluate this work, a large scale dataset with LDI representations is needed, which was not available at the time of the submission [22]. Section 3.2.1 describes the acquisition of the data which will be used for the purpose of this work.

3.2.1 Dataset generation

At the time of the submission of [22], there were no publicly available datasets containing LDI representations, suitable for deep learning purposes. The closest related [45] provided

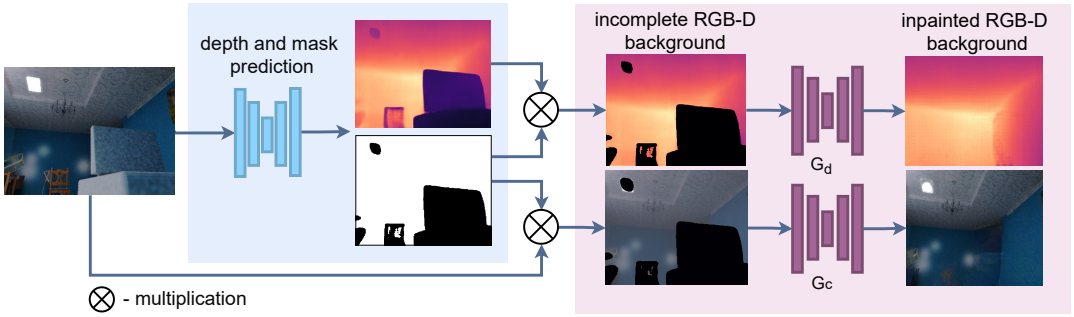


Figure 3.2. Overall two-layer method pipeline. (*Top*) A depth map and foreground mask are predicted simultaneously, via a fully-convolutional network. The RGB input and the predicted depth map are multiplied with the mask, to discard the foreground pixels. (*Bottom*) The partial RGB-D information is inpainted through a GAN architecture.

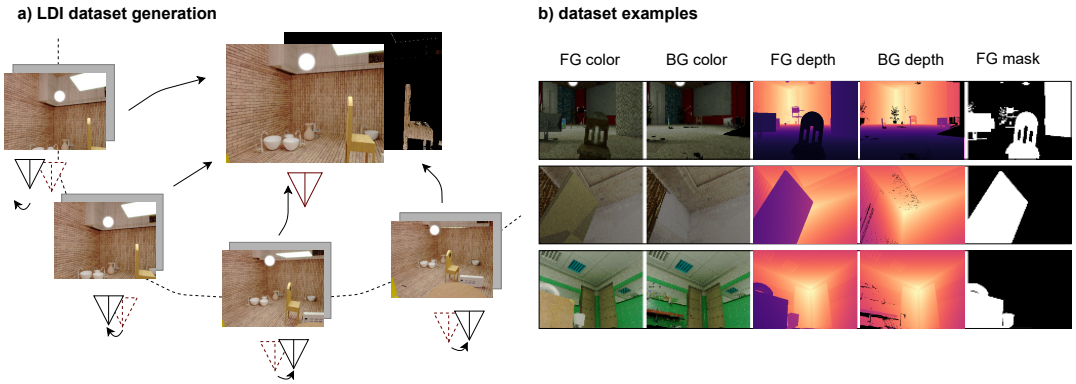


Figure 3.3. Dataset generation. *a)* LDI extraction from multiple views. *b)* Examples from the generated dataset. *FG:* Foreground, *i.e.* first RGB-D layer. *BG:* Background, including novel RGB-D pixels after dis-occlusion. [©2019 Pattern Recognition Letters]

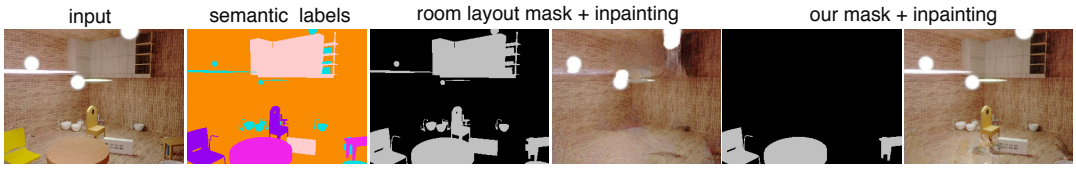


Figure 3.4. Illustration of semantic-based room layout separation vs. our adaptive foreground segmentation. [©2019 Pattern Recognition Letters]

a dataset of only 20 panoramic LDIs [1]. Hence, we decided to automatically generate an appropriate dataset for LDI learning, based on readily available RGB-D datasets that contain data sequences with associated camera poses.

We base our LDI dataset generation in an image-based rendering approach, that projects multiple available views of the scene onto a reference frame, Figure 3.3, a). This populates the reference frame with new RGB-D pixels, corresponding to occluded segments, *i.e.* that were not visible in the reference (source) view. For each frame we assume the camera pose is given. We used 40k RGB-D frames with the respective camera poses. To ensure continuity in the generated results – which facilitates learning with a CNN – we require

[1] http://visual.cs.ucl.ac.uk/pubs/casual3d/datasets.html

object instance annotations, to make sure that RGB-D values that correspond to the same object are stored in the same layer. Given these requirements on data modality, we base our dataset generation on a synthetic indoor dataset such as SceneNet [98], that consists of photorealistic environments.

The LDI extraction algorithm is as follows. We leverage a moving window of size $F = 20$ which extracts subsets of consecutive RGB-D frames corresponding to the same scene. We define the middle frame as reference view, f_{ref} and refer to the remaining ones as supportive frames. Then, we warp each pixel $\mathbf{u}$ of the supportive frames f_i into the reference view

$$\mathbf{u}_{\text{warped}} = \pi\left(\mathbf{T}_i^{\text{ref}} \pi^{-1}\left(\mathbf{u}_i\right)\right) \ , \tag{3.1}$$

where $\pi()$ denotes perspective projection and $\mathbf{T}_i^{\text{ref}}$ represents the transformation from f_i to f_{ref}. We use $\mathbf{u}_{\text{warped}}$ as the reference frame coordinates of the re-projected colors, instance labels and depth values. The moving window is incremented with a step of size 1, leading to an $F - 1$ overlap between consecutive frame subsets.

Along with pixel coordinates, the perspective projection gives the new depth values, corresponding to metric distance from a point in 3D to the reference camera plane. Since the SceneNet dataset [98] provides ray lengths r instead of depths d, the following conversion is applied to obtain the desired depth maps

$$d(\mathbf{u}) = \frac{r(\mathbf{u})}{||\mathbf{K}^{-1}\dot{\mathbf{u}}||_2}, \tag{3.2}$$

where $\mathbf{K}$ denotes the camera intrinsics and $\dot{\mathbf{u}}$ represents a pixel in homogeneous coordinates.

Once the depth values for the occluded background regions of the reference view are computed, *i.e.* warped from the support views, we want to select the pixels that will populate the background layer. Therefore we apply the following validity conditions:

1. The candidate depth value should be larger than the respective foreground (first layer) pixel at that location.
$$d_{\text{warped}}(\mathbf{u}_{\text{warped}}) > d_{\text{ref}}^{\text{FG}}(\mathbf{u}_{\text{warped}}) \tag{3.3}$$

2. The candidate pixel and the respective foreground pixel should have a different object id.
$$id_{\text{warped}}(\mathbf{u}_{\text{warped}}) \neq id_{\text{ref}}^{\text{FG}}(\mathbf{u}_{\text{warped}}) \tag{3.4}$$

3. The candidate pixel is a potential background pixel, only if the associated object instance does not occlude any other objects at any pixel location.

Finally, at each pixel location we keep in the background layer the pixel with the smallest warped depth among all valid candidates. We additionally extract a binary foreground mask, indicating pixels in the image whose reference instance label is found in the list of occluding

instances. In summary, the automatically acquired dataset consists of 40k samples, including a two-layer representation of RGB, depth, as well as a foreground segmentation mask. Figure 3.3, b) shows a few examples from the generated dataset. To obtain separate training and test data we utilize the split as proposed in SceneNet.

Note that the result of our method differs from a trivial room layout separation, which would instead give an "empty box" representation of the scene. In our particular application involving small vintage point perturbations, some object instances should practically be considered background, provided that they are not occluding other structures. Exemplary, in Figure 3.4 we expect to reveal segments of the occluded chair instead of just floor, during the dis-occlusion of the front table.

We empirically found out that the aforementioned LDI extraction algorithm does not work on real datasets like ScanNet [16], since not enough overlap is created between consecutive views and the dense segmentation annotations are not perfectly accurate. Nevertheless, while the ground truth generation relies on synthetic data, our method does generalize to the real domain as shown later in the experiments section.

3.2.2 Joint depth map and foreground mask prediction

The goal of this stage is to learn a mapping from an RGB image I to a conventional depth map d, which, combined with I, will represent the *first layer* of the LDI. We simultaneously aim to learn a guiding background mask m_{BG} that would later discard the foreground segments, thus indicating the regions of interest for the upcoming background completion. This can be interpreted as *implicit* learning of an adaptive threshold for the foreground-background separation, as a function of the distance from the camera, occlusion, as well as the structure continuity. For example, in Figure 3.2, wall regions that are closer to the camera compared to the brown chair, will still be categorized as background, since they are smoothly connected with the remaining part of the wall and do not occlude other objects.

For the purpose of depth and mask prediction, we employ a fully convolutional ResNet-50 architecture originally proposed by [76] for monocular depth prediction. The original network is extended with one additional up–projection block, which preserves the input resolution. We train our model on both tasks, separately and jointly. The latter shows superior performance for both tasks, which is intuitively justified as the underlying tasks are very related. The two tasks share all the network weights, except from the last layer, which branches out in two separate result layers. To supervise depth prediction we employ the reverse Huber loss $\mathcal{L}_{rH}(d, \hat{d})$, following [76], whereas the foreground segmentation is supervised through a L_2–norm $\mathcal{L}_2(m_{BG}, \hat{m}_{BG})$. We train jointly by combining the respective depth and segmentation losses with equal weight

$$\mathcal{L}_{dm} = \mathcal{L}_{rH}(d, \hat{d}) + \mathcal{L}_2(m_{BG}, \hat{m}_{BG}), \tag{3.5}$$

where d, m_{BG} denote the ground truth maps, while $\hat{\cdot}$ indicates predictions. Further, we apply an arbitrary threshold of 0.55 on the regressed masks to distinguish between background and

foreground. This threshold slightly favors classification as foreground to prevent undesired foreground segments from interfering with background structures during completion, while discarding background parts has a smaller negative effect on completion. Moreover, we observe that applying erosion on the resulting masks, further removes undesired false positives, *i.e.* wrong foreground predictions, typically located around the object boundaries.

3.2.3 RGB-D background inpainting

Predicting color and depth information in the occluded regions introduces additional ambiguities, as it is not easy to infer the unknown scene content and multiple solutions are possible. Hence, background inpainting is rather a hallucinatory task, that consists in creating plausible content conditioned on the available visible regions. Ideally, we want to generate realistic background images, that preserve texture details even on the more ambiguous dis-occluded parts. Therefore, we tackle the inpainting problem via a GAN-based approach, as GANs have shown promising results in generating realistic images, including RGB inpainting tasks [55, 58, 108].

Given a masked (incomplete) RGB-D input $(\tilde{I}_{BG}, \tilde{d}_{BG})$ we aim to obtain a complete RGB-D result (I_{BG}, d_{BG}), representing the background layer of a scene. Practically, $(\tilde{I}_{BG}, \tilde{d}_{BG})$ are obtained as $\tilde{I}_{BG} = m_{BG} I$, $\tilde{d}_{BG} = m_{BG} d$. We adopt a state-of-the-art model [58] and explore diverse ways of extending to the RGB-D case. For the generator G and the discriminator D we utilize similar architectures as in [58]. Written in its general form the adversarial loss leveraged in our inpainting GANs is

$$\mathcal{L}_a = \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_x[\log(1 - D(x, G(x)))] \tag{3.6}$$

Following a standard GAN formulation, G optimizes the following objective

$$\hat{G} = \min_G \max_D \mathcal{L}_a + \lambda_{\mathcal{L}_1} \mathcal{L}_1(y - G(x)). \tag{3.7}$$

The $\mathcal{L}_1$ loss measures the reconstruction error between the generated and ground truth samples.

On one hand, a shared RGB and depth completion learning could potentially strengthen their consistency. The representational difference between the two, however, promotes distinct learning. To examine the adverse reactions of these two motivations, we analyze various weight-sharing compositions.

Combined RGB-D completion In the combined approach, we employ a single generator architecture with four input and output channels (RGB-D). The discriminator receives eight channels, including the incomplete RGB-D prior as well as the generated or real RGB-D image. The objective function $\mathcal{L}_{RGB-D}$ follows Eq. 3.6 and 3.7, with $x = \tilde{I}_{BG} \oplus \tilde{d}_{BG}$ and $y = I_{BG} \oplus d_{BG}$, where $\oplus$ denotes concatenation.

Separate RGB and depth completion In the separate training approach we employ two separate generators and discriminators for color G_c, D_c and depth G_d, D_d, without any shared parameters between the respective RGB and depth parts. Following the same equations, here $\mathcal{L}_c$ with $x = \tilde{I}_{BG}$ and $y = I_{BG}$, as well as $\mathcal{L}_d$ with $x = \tilde{d}_{BG}$ and $y = d_{BG}$ are optimized independently.

Separate RGB and depth completion with pairing This final GAN model is built upon the latter approach and further combined with an additional multi-modal discriminator network which we call *pair discriminator* D_{pair}, that aims to encourage inter-domain consistency between RGB and depth. D_{pair} receives an RGB-D input, either from the ground truth data or the generator and distinguishes real RGB and depth *correspondences*. The respective generators G_d and G_c receive a signal from their separate discriminators as well as from the pair discriminator, thus optimizing an additional term

$$\mathcal{L}_{pair} = \mathbb{E}_{\tilde{I}^*_{BG}, I^*_{BG}}[\log D_{pair}(I_{BG}, d_{BG})] + \mathbb{E}_{\tilde{I}^*_{BG}}[\log(1 - D_{pair}(G_c(\tilde{I}_{BG}), G_d(\tilde{d}_{BG})))], \tag{3.8}$$

where I* denotes the respective tuple (I, d) for notation simplicity. The final $\hat{G}_c$ and $\hat{G}_d$ objectives then become

$$\hat{G}_c = \min_{G_c} \max_{D_c} \mathcal{L}_c + \lambda_{pair} \min_{G_c} \max_{D_{pair}} \mathcal{L}_{pair} + \lambda_{\mathcal{L}_1} \mathcal{L}_1(I_{BG}, G_c(\tilde{I}_{BG})) \tag{3.9}$$

$$\hat{G}_d = \min_{G_d} \max_{D_d} \mathcal{L}_d + \lambda_{pair} \min_{G_d} \max_{D_{pair}} \mathcal{L}_{pair} + \lambda_{\mathcal{L}_1} \mathcal{L}_1(d_{BG}, G_d(\tilde{d}_{BG})). \tag{3.10}$$

3.2.4 Implementation details

For the background mask erosion we used a cross–shaped structure with a size of 5×5 pixels. For the joint depth and mask prediction we used the Adam optimizer with a batch size of 8 and a learning rate of 0.001. All the inpainting discriminators adopt the C64-C128-C256-C512 architecture as proposed in [58], where C denotes a Convolution–BatchNorm–ReLU block followed by the number of filters. The loss weights are namely $\lambda_{pair} = 0.5$ and $\lambda_{L_1} = 100$. In all the inpanting models, we normalize the input color and depth images separately to $[-1, 1]$, and then set the inpainting regions to -2. The network then learns to identify image regions to be inpainted. For all the inpainting variants we train with a batch size of 1 and set the learning rate to 0.0002.

Though an interesting proof of concept for RGB-D inpainting on occluded regions, the naive two-layer method [22] is not ideal. Two layers are not sufficient to represent complex scenes, with multiple levels of occlusion. Moreover, the occluded areas of objects that fall on the foreground layer will not be covered. Hence, we introduced the following improvement in the method design, which will be explained in Section 3.3.

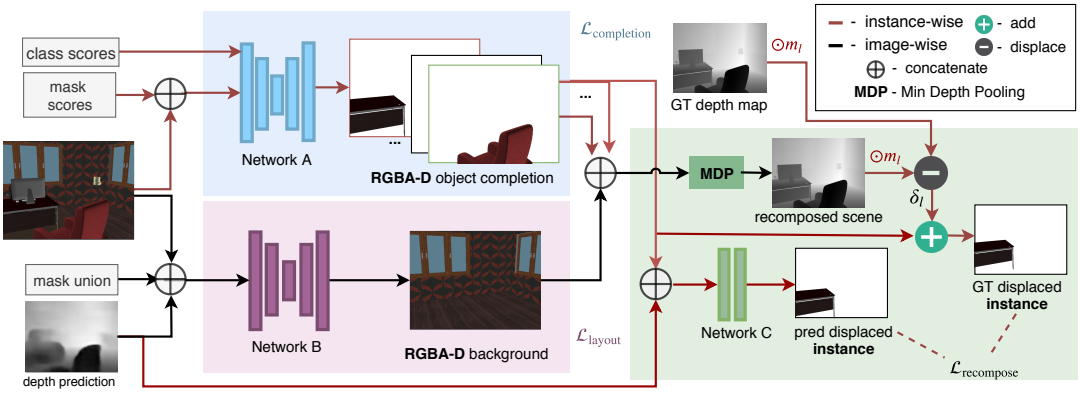


Figure 3.5. **OMLD scene layering framework.** *Left: Network A* (*top*) completes the occluded parts for each detected instance, resulting in an RGBA-D image. *Network B* (*bottom*) generates an RGBA-D representation for the layout (empty scene). *Right:* The outputs are concatenated and fed to the Minimum Depth Pooling (MDP) layer, that recomposes the scene an gives the first visible layer. The displacement of the recomposed first layer depth of every instance, from the ground truth depth is used in the re-composition loss to supervise *Network C* and produce the final result. [©2019 IEEE]

3.3 Object-Driven Multi-Layer Scene Decomposition (OMLD)

Since scenes come with varying levels of complexity, assuming a pre-determined number of layers to represent them is not ideal, as it limits their flexibility and representational power. In this section we explain the proposed multi-layer scene decomposition method, which is based on an object aware approach and therefore does not impose a restriction in the number of the generated LDI layers. Here the number of layers is controlled by the number of the detected object instances in the current scene. Starting with an RGB image I we formulate a framework that first decomposes the scene in the respective objects and layout (empty scene room). Then, each detected component is completed to its amodal visibility using two parallel networks for object completion (Section 3.3.2) and layout completion (Section 3.3.3). Finally, the set of generated amodal layers is fed to a re-composition component to enforce global scene consistency (Section 3.3.4). To supervise our framework, we need ground truth RGBA-D representations for each object $\{I_1^*, I_2^*, ..., I_N^*\}$ and the room layout I_{BG}^* in their amodal visibility. Section 3.3.1 describes the ground truth data generation procedure, based on layer-wise rendering from available 3D scenes.

3.3.1 Data generation

Our goal of object-wise layer inference implies the need for additional ground truth data for the supervision of the learned models, *i.e.* RGBA-D representations of every object and layout of the scene, which we acquire automatically from existing datasets. Different from the data generation approach for the two-layer method [22], here we employ a mesh-based scene rendering approach. The benefit of the 3D mesh is that it captures all the available information in the scene, while the former image-based rendering only captures information which is casually available in the given set of consecutive image frames. The rendering algorithm works as follows. For every frame, each instance which is visible in that view is rendered

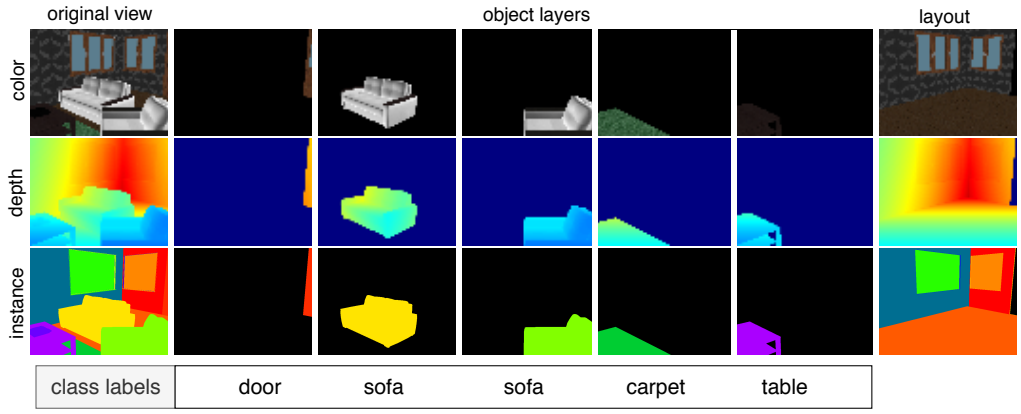
separately, in the form of an RGBA color image, depth map as well as an object category label. Thereby, we utilize the dense semantic and instance annotations, accompanying the 3D meshes of the respective datasets, to extract the vertices of each visible object in every view frame. Structural elements are identified by their semantic category (floor, wall, ceiling, window) and grouped together to compose the layout layer representing the empty room. In case of multiple structural instances falling on the same pixel, the point closer to the camera, *i.e.* lowest depth, is kept. Note that we discard all instances that were not originally visible in a certain view. For instance, an object located behind the enclosing wall of the currently visible room is not desired as part of the compositional layers of that view. The proposed semantic-aware rendering offers the additional advantage – compared to [22, 139] – that it enables learning of class specific features, which might turn helpful in regressing more realistic objects in the synthesized LDI layers.

In this work we render from two different 3D datasets, namely the synthetic SunCG [131] and the real Stanford 2D-3D-S [3]. Both datasets contain scene meshes together with 2D modalities (color, depth, instances, semantics). The latter suffers from the presence of holes and missing surface parts –a typical real-world mesh nuisance – which in our task results in incomplete object renderings. Therefore, we leverage a post-processing step, to filter the rendered layered images, based on the amount of overlap between layers. Thus, when the number of overlapping pixels among all the layers surpasses an empirically chosen threshold, the respective layered representation contains enough novel information in case of dis-occlusion. The purpose of utilizing a synthetic dataset [131] is for a fair evaluation, given pixel-perfect ground truth, while the real-world dataset [3] demonstrates applicability in a real setting. We show in Figure 3.6 two examples from the automatically generated datasets.

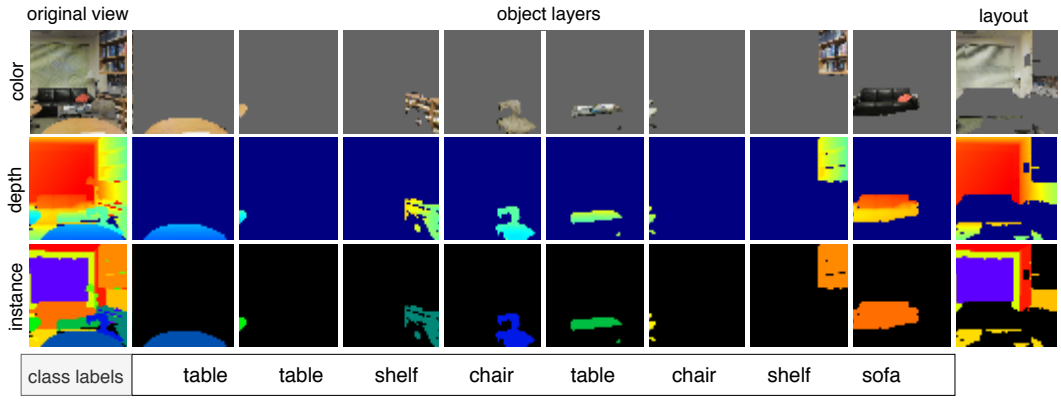
The generated object and layout layers can be easily arranged in an LDI representation, using the depth maps to sort the layers at every pixel location.

3.3.2 Object completion

The objective of the object completion network (Network A, Figure 3.5) is to establish a mapping from the modal RGB perception of an object, *i.e.* occluded object in the input image I to its amodal RGBA-D perception, *i.e.* complete representation I_i^*. We additionally utilize a binary modal mask $\hat{m}_i$ as a prior for each object, as in other ambiguous tasks [23, 27]. Additionally, we incorporate semantic class labels, to encourage class-aware learning. Network A is thus composed of two branches (Figure 3.7), which receive namely the input RGB image concatenated with the amodal mask of an object $\tilde{I}_i = I \oplus \hat{m}_i$, as well as the associated class score $\hat{c}_i$; and predicts the amodal RGBA-D representation of that object, amounting to a five-channel result $\hat{I}_i^* = (\hat{I}_i, \hat{d}_i)$. This object completion model is applied instance-wise, for each available modal mask. The architecture details are provided in Section 3.3.5. Note that different from similar works in object completion [23], we feed whole images in Network A instead of regions of interest (RoIs) focused on each instance. Though RoI cropping leads in general to more efficient computation, we argue that, first, by cropping and resizing one alters the focal length of an image, which weakens depth perception, as it



(a) Based on SunCG



(b) Based on Stanford 2D-3D-S

Figure 3.6. **Illustration of the automatically acquired datasets.** For every view frame in the respective datasets, we provide the RGBA, depth, instance segmentation and class categories. All modalities are available for full-image content, object-wise layers as well as the layout.

lessens global context and hinders the understanding for object extent and scale. Second, fixed-size RoIs limit the generation resolution, which mostly affects the texture details of large objects.

Taking into consideration real world input conditions where ground truth modal masks are not given, we practically leverage predicted masks from Mask R-CNN [42], both at training and inference time. At training time, to match each ground truth instance mask with the closest mask from the set of Mask R-CNN predictions, we employ intersection-over-union (IoU). We however discard matches that have an $\mathrm{IoU} < 0.3$, to avoid wrong correspondences. We then train the object completion network using the remaining instances, for which we have a (valid) ground truth – prediction pair. We incorporate a L1 loss objective on the amodal RGBA-D predictions $\hat{I}_i^*$, which is weighted by a relevance map γ

$$\mathcal{L}_{\text{completion}} = ||\gamma(I_i^* - \hat{I}_i^*)||_1. \tag{3.11}$$

The need for a dense weighting γ arises from the fact that we want to avoid learning shortcuts, as different image regions impose different difficulty level for the task at hand. For instance,

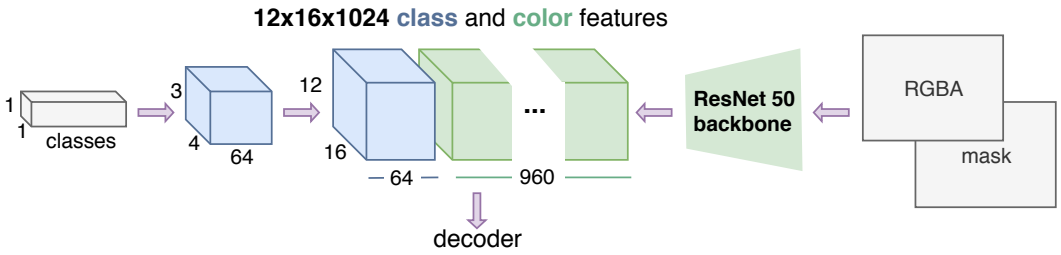


Figure 3.7. **Overview of the proposed object completion encoder architecture.** Class probabilities branch *(left)* and image branch *(right)* are concatenated along channels in the bottleneck layer. [©2019 IEEE]

the network could learn to just copy information from the input to the output, considering that a considerable fraction of the image contains visible object regions. Additionally, it can easily learn to generate zeros since an object's extent typically spans a small fraction of the whole image. Hence, we want to set a higher error influence in pixels close to the object appearance, were dis-occlusions are expected. Thus we set $\gamma = 0.7$ in the *visible area* of the object in the original image including *a neighbourhood* (by applying a dilation of size 31×31 in the ground truth modal mask), $\gamma = 1.5$ in the *occluded regions* of that instance, and $\gamma = 0.2$ otherwise. At inference time, each mask and class label prediction from Mask R-CNN [42] is fed to the object completion network together with the input RGB image, to obtain each amodal RGBA-D object layer.

The task of object completion from a single RGB input imposes a set of concurrent challenges. First, copying the modal area might represent already a relatively good shortcut solution, though the occluded areas are not properly learned. Second, the guiding modal masks are not perfect around the edges, therefore the network has to differentiate between the object instances along boundaries. Therefore, an additional pre-training step is employed, to encourage the network in learning features of object distributions in complete visibility and generate plausible outputs. Thereby prior to learning the original completion task, we train Network A as an Auto-Encoder $\hat{I}_i^* = f_{auto}(I_i^*)$ (*i.e.* in an unsupervised way) on amodal RGBA-D object representations

$$\mathcal{L}_{auto} = ||I_i^* - \hat{I}_i^*||_1. \tag{3.12}$$

Here we utilize only object instances which do not touch the image borders to guarantee full object visibility. Further, we freeze the decoder weights (including the bottleneck layer) and re-train the encoder – this time for object completion – using the supervision scheme from Eq. (3.11). This Auto-Encoder based initialization helps the completion task by encouraging the partially occluded objects to share the same latent space as their respective complete RGBA-D representation.

3.3.3 Layout prediction

The goal of the layout prediction network (Network B in Figure 3.5) is to learn a mapping from the input RGB image I to the corresponding RGB-D scene layout I_{BG}^*, *i.e.* the empty room of the scene at hand. In addition to I, we utilize a standard predicted depth map $\hat{d}$ as a

prior, given that empirical findings suggested that this gives considerable improvement on the depth layout prediction. Note that such input did however hinder the performance on the object completion counterpart, which we relate with the smeared out object boundaries on predicted depth maps at the time of publication. Further, the layout prediction model would benefit from a binary mask indicating the background/foreground areas. We estimate such background mask as the union of the full set of predicted object modal masks $\hat{m}_{BG} = \bigcup_{i=1}^{n} \hat{m}_i$. Thus our model receives an RGBA image I, concatenated with a learned depth map $\hat{d}$ (in our experiments we leverage [76]), as well as the obtained background mask $\hat{m}_{BG}$. Different from the background inpainting in PBO (Section 3.2.3), here we do not occlude the non-structural regions while feeding the input image. Since the predicted masks are noisy variants of the true masks, they can potentially leave out relevant segments from the input.

We supervise the layout prediction network via an adversarial loss [38] to encourage generation of realistic layouts. Moreover, we put emphasis on the consistency of the generated edges, such that the room contour – an interesting property of the layout – does not get overlooked by the network. A similar motivation has been discussed in a depth completion task [169], where occlusion boundaries are estimated as an intermediate step to improve the final result. Instead of explicitly acquiring and supervising such boundaries [169], we utilize a perceptual loss term $\mathcal{L}_p$ [64]. The synthesized and ground truth layouts are separately fed to a VGG-16 [129] network, trained on a classification task on ImageNet [18]. Then, the perceptual loss $\mathcal{L}_1$ computes the distance between the respective VGG-16 feature maps from ground truth and predictions. Here we leverage the features at the end of the first block, given that it captures the edges as desired. Thus, the total loss objective becomes

$$\mathcal{L}_{layout} = \lambda_r \mathcal{L}_r + \lambda_p \mathcal{L}_p + \min_G \max_D (\mathcal{L}_a) \tag{3.13}$$

$$\mathcal{L}_r = ||I_{BG} - \hat{I}_{BG}||_1 + ||d_{BG} - \hat{d}_{BG}||_1 \tag{3.14}$$

$$\mathcal{L}_p = ||\phi_1(I_{BG}) - \phi_1(\hat{I}_{BG})||_1 \tag{3.15}$$

$$\mathcal{L}_a = \mathbb{E}_{\tilde{I}_{BG}, I_{BG}}[\log D(\tilde{I}_{BG}, I_{BG})] + \mathbb{E}_{\tilde{I}_{BG}}[\log(1 - D(\tilde{I}_{BG}, G(\tilde{I}_{BG})))], \tag{3.16}$$

where $\tilde{I}_{BG} = I \oplus \hat{m}_{BG} \oplus \hat{d}$, $\hat{I}_{BG}$, $\hat{d}_{BG}$ denote the output color and depth respectively, and ϕ_1 is the resulting feature map of the first VGG-16 block.

Network B is essentially implemented as a U-Net [118] based architecture with skip connections, whose details are provided in Section 3.3.5.

3.3.4 Image re-composition

So far, each layer is inferred independently, without awareness of the other object instances or the enclosing room layout. This becomes particularly relevant as we try to re-compose the entire scene. The goal of the third component in the OMLD model, is thus to improve the global depth consistency of all regressed scene parts (objects and layout). The algorithm first performs concatenation of all the generated layers and, at each pixel location, extracts

the RGBA-D value from the layer with the minimum depth, *i.e.* essentially a z-buffer. We name this step *minimum depth pooling* (MDP). The result of MDP is an RGB image – ideally identical to the original input – in addition to the corresponding visible depth map, as well as an index map i_{map} that represents the argmin of depth, which indicates the instance index from where the information in that pixel location is copied. Next we explain how we use the obtained information from MDP to impose structural coherency between the generated multi-layer representation and the original input image.

Since learned models for monocular depth prediction fairly learn a global understanding for depth, which is aware of the whole image content, we use a conventional inferred depth map $\hat{d}$ as a prior for the layer sorting task. For each layer l, we extract the region in $\hat{d}$ in which l is the front instance, using a binary mask m_l, which is one if $i_{map} = l$, and zero otherwise. We then incorporate a re-composition block (*Network C*), that consumes the predicted instance depth $\hat{d}_l$, concatenated with the masked conventional depth prediction $\hat{d} \odot m_l$ from [76]. Then we define a set of depth displacements δ_l, as the distance between the mean ground truth d_l and the mean predicted $\hat{d}_l$ layer depth (over m_l)

$$\delta_l = \frac{\sum m_l \odot d_l}{\sum m_l} - \frac{\sum m_l \odot \hat{d}_l}{\sum m_l}, \tag{3.17}$$

where $\odot$ is the element-wise multiplication. Further, we obtain a pseudo ground truth depth for each instance l, using the originally predicted depth layer as $d_{\delta,l} = \hat{d}_l + \delta_l$ and optimize the following objective

$$\mathcal{L}_{\text{recompose}} = ||d_{\delta,l} - \hat{d}_l||_1. \tag{3.18}$$

This shifts the predicted depth to reach global consistency, while not affecting the local characteristics of each object.

Our experiments suggest that the proposed MDP-driven re-composition loss improves both the modal areas of the objects, and the occluded parts, given that it preserves the shape of the objects, while allowing the entire layer to shift towards the right direction.

3.3.5 Implementation details

This section provides the implementation details of the OMLD model.

Network A The object completion network accepts two inputs, namely a concatenation of the input RGB image with the modal mask score, and a vector of class scores, as predicted from Mask R-CNN [42], Figure 3.7. Network A is essentially a ResNet-50 [43] backbone, where the original fully-connected layer is discarded. We append a convolution layer instead with $\text{out}_{\text{channels}} = 960$ channels. The class branch consists of two deconvolution layers of 64 channels each applied consecutively on the class scores, which is a feature vector with size equals to the number of classes. The results of both branches are then concatenated along

the channel dimension, followed by *layer normalization* [6]. Further, the resulting feature is processed via a decoder architecture which contains five up-projection layers [76]. The network used for the Auto-Encoder pre-training has the same architecture as the object completion network, however, given the partially different input modalities, we learn the completion encoder from scratch instead of fine-tuning its Auto-Encoder counterpart.

Network B The layout network follows a U-Net [118] composition, similar to [58]. The generator G is an architecture with skip connections which contains seven down-sampling (convolution blocks) and up-sampling level (deconvolution blocks), each layer having a stride of two. The number of output channels starts at 64 and is further doubled after each convolution layer. Similarly, each deconvolution layer halves the feature map channels. After each deconvolution, the output feature map is concatenated with the feature map in the encoder, which has the same resolution. The discriminator D contains six convolution blocks and a final fully-connected layer. The discriminator feature maps contain namely 64, 128, 256, 512, 512 and 512 channels. All blocks in G and D contain a (de)convolution layer followed by batch normalization and leaky ReLU activation. The loss weights used for training are $\lambda_r = 100$ and $\lambda_p = 25$.

Network C The re-composition network consists of three convolutional layers with filter size 3×3, each followed by a ReLU activation.

Zero-padding on image borders As LDIs are mainly intended for view synthesis, during viewpoint change the novel image contains empty regions on the borders, for content that was beyond the visible frame from the vintage point. To provide some additional context in the original view, we add zero padding to our input images prior to being processed by the network. Hence, the original frame is spanned by predicting the originally padded surroundings. In our experiments we use horizontal padding bands of 16 pixels and vertical bands of 12 pixels.

Training We train Network A, B and C separately, using the Adam Optimizer [70] with a learning rate of namely $1 \cdot 10^{-4}$ for Network A, $2 \cdot 10^{-3}$ for Network B and $1 \cdot 10^{-3}$ for Network C. The batch size is 4 (resolution 384×512) for the experiments on SunCG and 8 (resolution 256×256) for Stanford 2D-3D-S. The whole training takes 2 to 3 days on an Nivida RTX GPU.

3.4 Evaluation

This section presents the evaluation protocol and the performed experiments to assess the performance of the proposed methods, PBO and OMLD. We report qualitative and quantitative evaluation in three different tasks. First we evaluate the direct outcome of the methods, *i.e.* the image and depth layers. Second, we evaluate a novel view synthesis application after warping the generated layers in a different view. Additionally, we introduce the task of image manipulation from an LDI, in which we can remove components of choice

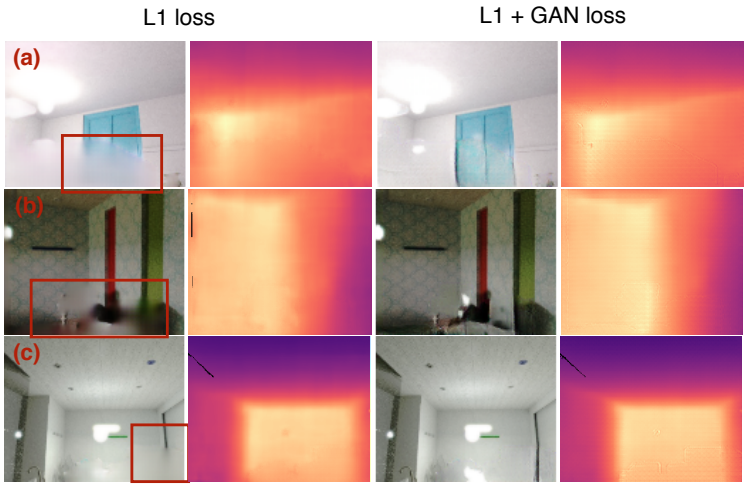


Figure 3.8. Qualitative comparison on the generation performance of the pixel-wise L_1 loss (left) and the adversarial variant (GAN loss + L_1) (right). *Red box:* Regions of interest. [©2019 Pattern Recognition Letters]

from the original scene, exploiting the semantic compositional nature of our approaches. Note that the latter task is not possible with other LDI generation works.

PBO We evaluated the two-layer model on the SceneNet [98] and NYU depth v2 [102] datasets. For SceneNet, we use our ground truth background data described in Section 3.2.1, therefore we present both qualitative and quantitative results. In absence of ground truth data in the real domain, we only show qualitative results on NYU. We compare against two baselines for LDI generation [139] and view synthesis [139, 178].

OMLD We evaluated the object-driven model on two public benchmark datasets: SunCG [131] and Stanford 2D-3D-S [3], using the generated LDI dataset explained in Section 3.3.1. To obtain dense layers for the sparse object layers, we merge the output objects into a layered representation, in accordance with the original LDI idea [125]. In each pixel, the first layer represents the first visible point along the ray-line, the second layer relates to the next visible surface point and so on. The merging is done by an extended version of MDP, which sorts the depth of the object-wise layers, instead of simply returning the minimum. We evaluate OMLD against two baselines, namely the proposed PBO and Tulsiani *et al.* [139].

3.4.1 Conventional depth and background mask

We train the joint depth map and foreground segmentation prediction task through a subset of around 30k RGB-D samples from the train split of the SceneNet dataset. Further, we test on around 500 images, from the *validation* split. For the depth prediction task we obtain an absolute relative error (REL) of 0.184. The mask prediction task is instead evaluated using intersection over union (IoU), which leads to an accuracy of 0.71 for foreground and 0.93 for background.

Method	Region	RGB metrics			Depth error	
		SSIM ↑	RMSE ↓	PSNR ↑	REL ↓	RMSE ↓
Tulsiani *et al.* [139]	*whole*	0.781	30.76	18.37	0.511	3.731
	inpainted	–	67.33	11.57	0.606	3.893
PBO combined	*whole*	0.914	**20.74**	**21.79**	0.156	0.574
	inpainted	–	**56.75**	**13.05**	0.197	0.704
PBO separate	*whole*	0.918	21.78	21.37	0.151	0.555
	inpainted	–	60.78	12.46	0.175	0.633
PBO separate with D_{pair}	*whole*	**0.919**	21.76	21.38	**0.149**	**0.549**
	inpainted	–	57.94	12.87	**0.172**	**0.622**

Table 3.1. Comparison of our GAN versions with [139], on SceneNet. Inpainting applied on *learned background masks and depths*. [©2019 Pattern Recognition Letters]

3.4.2 Background inpainting

Quantitative results on the background layer generation for PBO are reported in Tables 3.1 and 3.2. In the former, the final results are products of our full pipeline that consumes a single RGB input. Here the first layer depth and the background mask are predicted as described in Section 3.2.2. In the latter, we feed ground truth depth maps and background masks in the background inpainting model. We present a comparison against [139] as the only LDI generation work at the time of publication. We train this method in SceneNet, on the same splits as ours, utilizing the official code repository from the authors. In addition, we ablate the different weight-sharing variants introduced in Section 3.2.3. For this purpose, to evaluate the generated depth maps we compute two common metrics used in depth prediction works, such as the relative error (REL) and root mean square error (RMSE). To assess texture image generation performance, in addition to the RMSE error we utilize structural similarity index (SSIM) and peak signal to noise ratio (PSNR). The errors are measured in two different settings, one for the entire image, and the other on the inpainted background regions only. As Table 3.1 shows, we outperform [139] in all metrics. On the first layer depth, their relative error is 0.395, so considerably higher compared to PBO. Note that [139] leverage a self-supervised LDI prediction approach, via view synthesis, which makes depth estimation a harder task. This explains the generally large error gap for depth.

The ablation study in Tables 3.1 and 3.2 shows that the *combined RGB-D inpainting* variant performs the poorest in terms of depth prediction. In contrast, the accuracy of the RGB inpainting has a slight advantage compared to the other methods. We believe that the RGB counterpart is dominating the learning signal. Textural features from the color domain could induce unnecessary gradients for the depth prediction task, thus having a negative impact on depth regression performance. Moreover, one could argue that the color inpainting benefits from depth information, which provides hints for the separation between different structures.

Method	Region	RGB metrics			Depth error	
		SSIM ↑	RMSE ↓	PSNR ↑	REL ↓	RMSE ↓
PBO combined	*whole*	0.891	**19.45**	**22.35**	0.044	0.189
	inpainted	–	**51.32**	**13.92**	0.090	0.349
PBO separate	*whole*	0.900	20.09	22.07	0.018	0.100
	inpainted	–	53.98	13.49	**0.041**	**0.193**
PBO separate with D_{pair}	*whole*	**0.903**	19.76	22.22	**0.017**	**0.095**
	inpainted	–	52.70	13.69	**0.041**	0.197

Table 3.2. Analysis of our GAN versions, on the SceneNet dataset [98]. Inpainting module applied on *ground truth masks and depths*. [©2019 Pattern Recognition Letters]

The *separate RGB and depth* approaches, with and without pairing lead to a clearly better performance for background depth inpainting, particularly on the inpainted background region, which is of high interest. Further, the pairing loss $\mathcal{L}_{pair}$ brings an improvement in the depth inpainting task, particularly noticeable when learning from predicted depths and masks (Table 3.1). Figure 3.10 demonstrates the ablation of the pair discriminator D_{pair} visually. One can observe that the respective RGB and depth counterparts in the inpainted background are better aligned in the variant that leverages D_{pair}.

Figure 3.8 illustrates the qualitative difference of using a GAN loss term in addition to the pixel-wise reconstruction loss L_1. Notably, the model based purely on L_1 loss generates relatively accurate images, while it fails to capture local and global real world visual patterns, such as realistic object shapes (*e.g.* the door in image a), edge sharpness (example a and c), and fine texture details (b). Also the depth maps tent to be slightly sharper in the combined L_1 and GAN variant. This motivates the choice of using an adversarial approach in this work.

As expected the background inpainting performance is better when a more accurate depth (here ground truth) is available for the first layer (Table 3.2 compared to Table 3.1). Notably, the 2^{nd} layer relative error on Table 3.2 is in a very low range, while the relative errors in Table 3.1 are comparable to the inherent first-layer depth prediction error from Section 3.4.1. Moreover, the relative errors from the depth inpainting results are even lower than those of first-layer depth map prediction. While conventional depth map predictors usually learn a decent absolute scale, difficult regions such as object borders tend to be smeared, which introduces additional errors, compared to the simple object-less background layer. Interestingly, the proposed PBO method is a flexible asset that can be easily integrated into a regular depth prediction model, without affecting the accuracy. In particular, as the quality of CNN-based depth prediction improves, the LDI generation becomes more accurate, without major changes in the proposed method.

Figure 3.9 demonstrates qualitative results of the background inpainting task on SceneNet. Here the only input is the RGB image, *i.e.* the first-layer depths and masks are predicted from

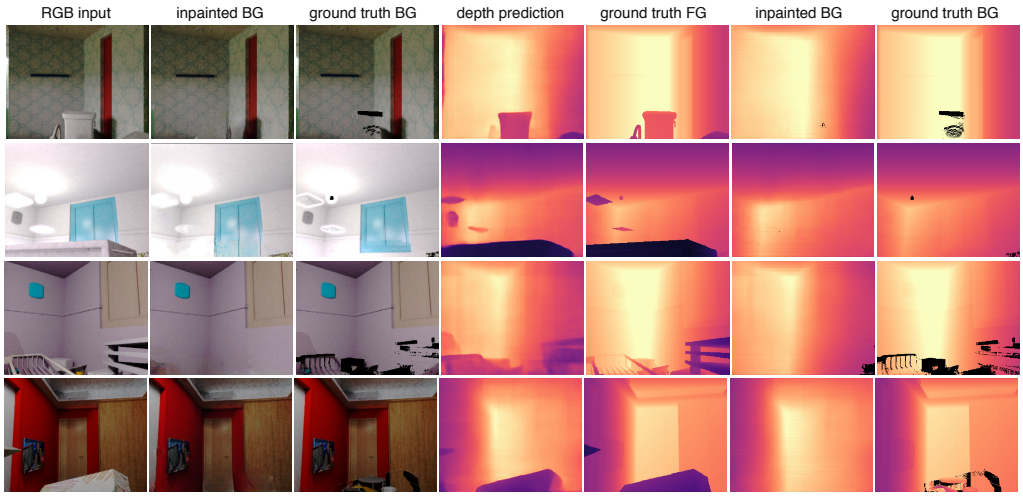


Figure 3.9. Qualitative examples of foreground removal and RGB-D background inpainting, evaluated on SceneNet [98], with the respective ground truth. *Black*: invalid pixels. [©2019 Pattern Recognition Letters]

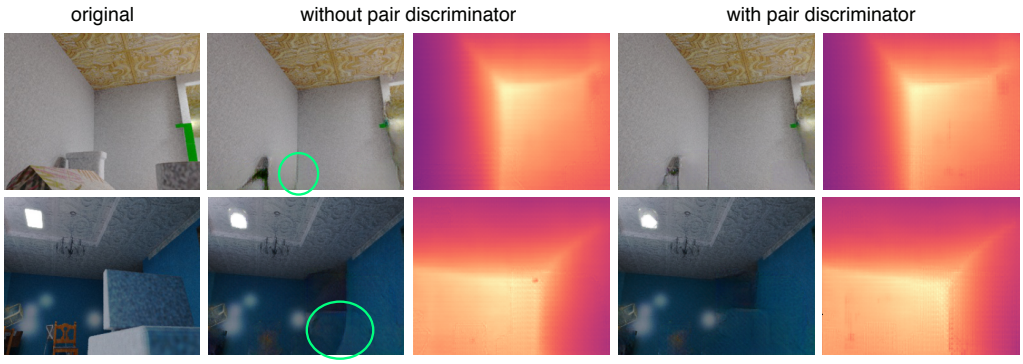


Figure 3.10. Qualitative results of the background completion RGB-D alignment. *(Left)* Original input. *(Center)* Result without pair discriminator. *(Right)* Result with pair discriminator.

the first PBO stage. Though the background ground truth is not always available, as some regions have not been visible from any viewpoint during warping, the generated images are always complete. Interestingly, the model can occasionally hallucinate a different object from the ground truth, which in turn is equally plausible considering the occlusion ambiguity, *e.g.* second example, our model generated a door as opposed to the ground truth window.

Figure 3.11 provides qualitative results on the mask prediction, depth estimation and RGB-D background inpainting on the real NYU depth v2 dataset [102], to illustrate the method's capability to generalize to real environments. Here the ground truth background is not available. Nevertheless, one can judge on the inpainting plausibility by observing the generated images. To bridge the domain gap between synthetic and real data, we fine-tune the weights learned on SceneNet with classic inpainting and segmentation tasks. As in a classic inpainting problem, we apply random occluding masks on the NYU images to mimic an occlusion scenario and train the GAN to reconstruct these missing regions. Similarly, the background mask prediction task is fine-tuned on the real domain, using NYU semantic labels to separate foreground (non-structural categories) from background (structural categories).

Figure 3.11. Qualitative examples of background mask prediction and RGB-D background inpainting on NYU depth v2 [102]. [©2019 Pattern Recognition Letters]

For the first-layer depth prediction task, we train the respective model (pre-trained on ImageNet) fully on NYU, as depth data is readily available in this dataset [102].

3.4.3 Layered representation

We evaluate the OMLD method against state-of-the-art work in monocular LDI generation, which at the time of publication were PBO and Tulsiani *et al.* [139]. The comparison against PBO evaluates the relevance of the multi-layer representation, which assumes multiple levels of occlusion in the image. Furthermore, the comparison with Tulsiani *et al.* [139] confirms the advantage of exploiting a direct supervision for the LDI generation. For a fair comparison against the two-layer baselines, we evaluate OMLD using our first two layers in the quantitative and qualitative evaluation. Additionally, we report the results on all layers separately on a plot figure (Figure 3.15). Further ablation and qualitative evaluation on intermediate results can be found in appendix B. We evaluate our results on two metrics,

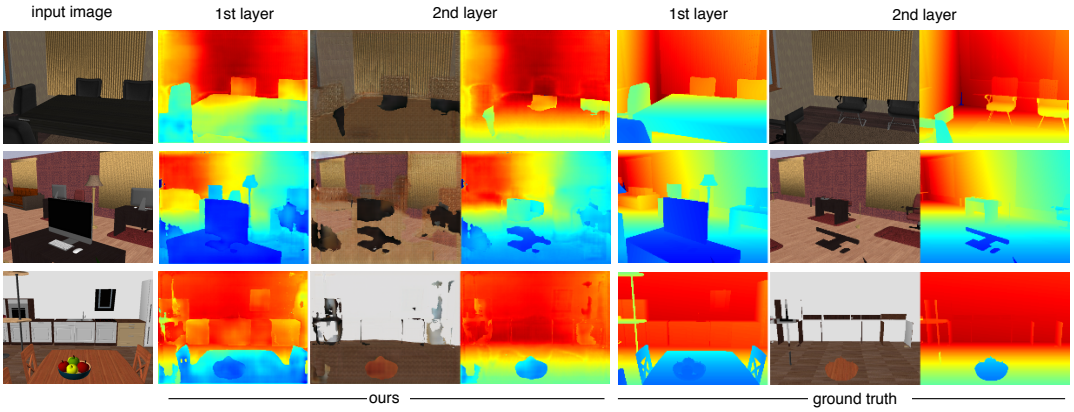


Figure 3.12. **LDI prediction results on SunCG.** *Left:* The input color image. *Center:* Our predictions for the first two layers, obtained after sorting the object-wise layers. *Right:* Ground truth, as extracted from the mesh-based rendering.[©2019 IEEE]

Mean Pixel Error (MPE) and Root Mean Square Error (RMSE). The measurements for each layer are done separately, since the difficulty is expected to depend on the layer index.

In these experiments, we use Mask R-CNN [42] predictions for the mask and class scores as input to our framework. For Stanford 2D-3D-S, we employ a network trained on the MS-COCO dataset [88]. For our experiments on SunCG, we finetune the network pre-trained on MS-COCO, using the NYU 40 class categories. As for the input depth predictions, we use the model from Laina *et al*. [76], respectively trained on SunCG and Stanford 2D-3D-S. We chose [42] and [76] as common baselines with available code. We also make this choice for fairness of comparison, since in PBO [22] we also use [76] to predict the first layer depth. However, OMLD is practically agnostic to the choice of these models.

SunCG All experiments on the SunCG dataset are carried out on the same splits, with 11k images on the training set and 2k on the test set. The quantitative results are shown in Table 3.3. As the first color image represents the input itself, we only report depth for the first layer. OMLD clearly outperforms [139] and PBO in all metrics, considering both the depth and color component. In addition, we perform an ablation on the most relevant components of OMLD, such as semantic-awareness, the perceptual loss $\mathcal{L}_{\mathrm{p}}$, and the re-composition block. One can observe an improvement from adding each component, with a more noticeable effect on depth prediction.

We refer the reader to Figure 3.13 for a visual comparison between the methods, with a particular qualitative difference. As OMLD learns the depths instance-wise, it overcomes the problem of smeared object boundaries, which is a common nuisance of many CNN-based depth estimators. Sharp object edges are a desired property in view synthesis in particular, as smooth edges lead to very noticeable shape deformations in the warped images. More qualitative examples generated by OMLD can be seen in Figure 3.12.

Stanford 2D-3D-S From all the acquired data samples from 2D-3D-S, we extracted 14k images with considerable ground truth coverage, namely 13k for the training set and 1k for the test set. We follow one of the splits from the original 2D-3D-S paper [3] (area 1,2,3,4,6

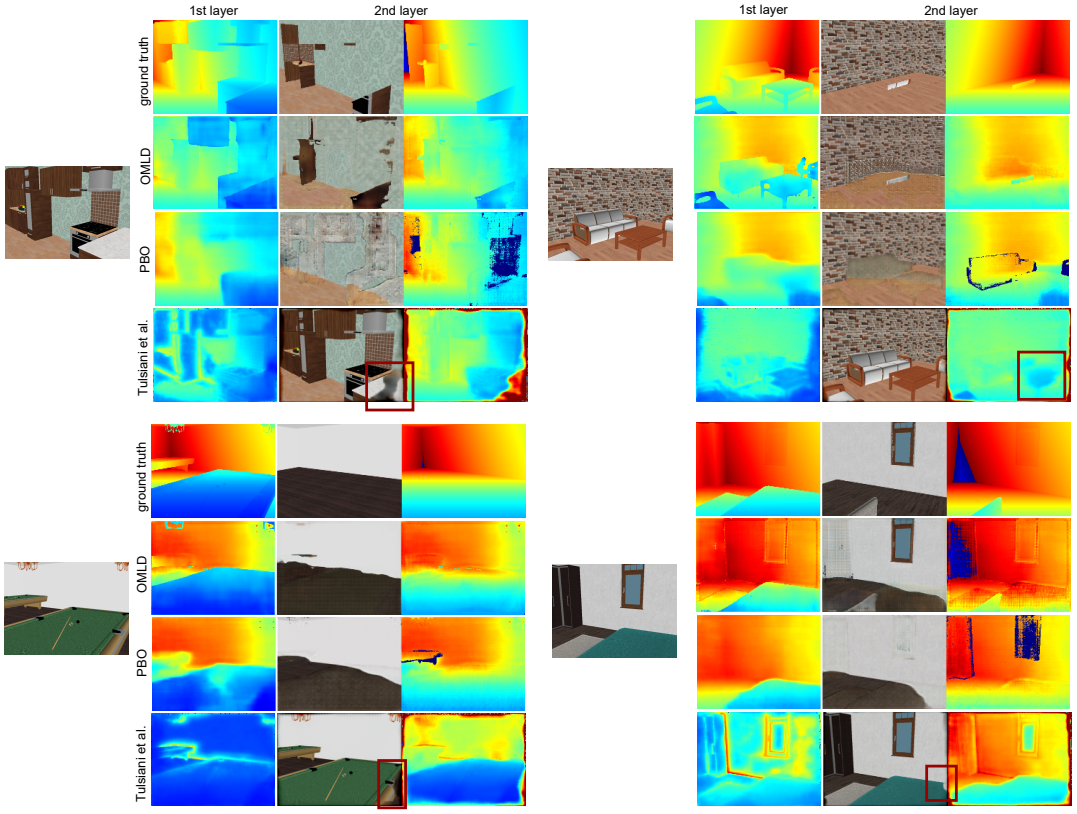


Figure 3.13. **Comparison of LDI generation results on SunCG.** For each example, *Left:* The input color image. *Right:* From top to bottom - ground truth, two-layer predictions of OMLD, PBO and Tulsiani *et al.* [139] for the first two layers.

Method	1st layer depth		2nd layer depth		2nd layer RGB	
	MPE	RMSE	MPE	RMSE	MPE	RMSE
Tulsiani *et al.* [139]	1.174	1.687	1.582	2.873	72.70	91.51
PBO	0.511	0.832	1.139	1.848	48.57	76.98
OMLD, baseline (w/o classes)	0.551	0.879	0.687	1.120	43.97	66.51
+ class scores	0.508	0.793	0.700	1.090	44.50	65.70
+ $\mathcal{L}_{\text{p}}$	0.496	0.800	0.657	1.095	43.92	66.48
+ $\mathcal{L}_{\text{recompose}}$	**0.473**	**0.767**	**0.641**	**1.071**	**43.12**	**65.66**

Table 3.3. **Evaluation of LDI prediction on SunCG** for the first two layers of depth and the 2nd layer of RGB. We outperform the baselines. The errors are measured for color range $0-255$ and depth in meters.[©2019 IEEE]

vs. area 5a,b). We pre-trained OMLD on SunCG and fine-tuned on Stanford 2D-3D-S. We infer masks and semantic labels for 2D-3D-S by running a Mask R-CNN model trained on COCO. To map the semantic categories across datasets, we convert the COCO classes to NYU-40 (as in our SunCG models). For the 2D-3D-S layout training we had to disable the

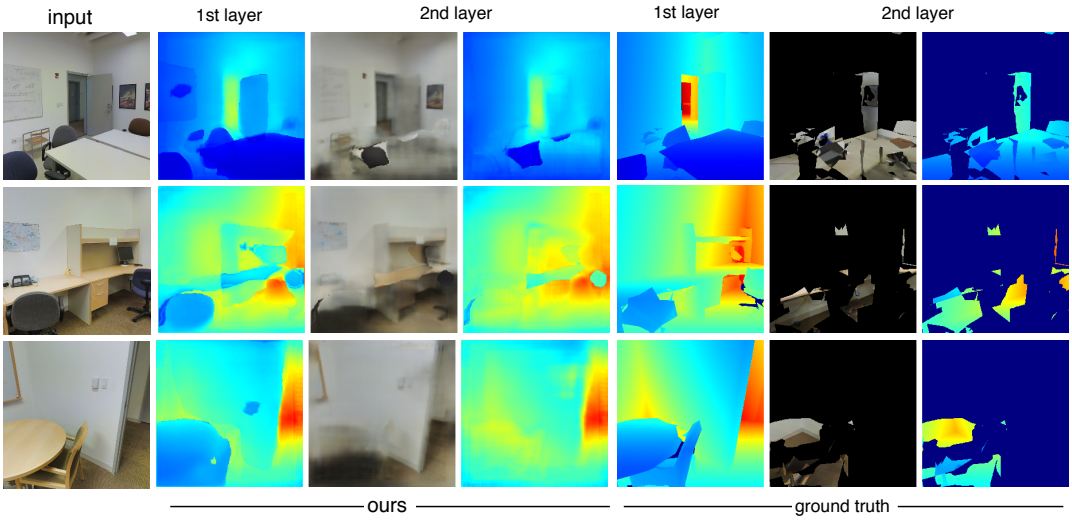


Figure 3.14. **LDI prediction results on Stanford 2D-3D-S.** *Left:* The input color image. *Center:* Our predictions for the first two layers, obtained after sorting the object-wise layers. *Right:* Ground truth, as extracted from the mesh-based rendering. Black in the color images and dark blue in the depth maps indicates information holes.[©2019 IEEE]

Method	1st layer depth		2nd layer depth		2nd layer RGB	
	MPE	RMSE	MPE	RMSE	MPE	RMSE
Tulsiani *et al*. [139]	0.805	1.088	0.954	1.230	57.42	72.65
PBO	**0.456**	**0.676**	0.830	1.193	42.92	55.87
OMLD w/o $\mathcal{L}_{\text{recompose}}$	0.509	0.764	0.692	0.993	42.57	55.07
OMLD	0.469	0.695	**0.688**	**0.987**	**42.45**	**54.92**

Table 3.4. **Evaluation of LDI prediction on Stanford 2D-3D-S.** LDI predictions for the first two layers of depth and 2nd layer of RGB. The errors are measured for color range $0-255$ and depth in meters.[©2019 IEEE]

GAN loss, since the network was generating sparse layouts, trying to mimic an undesired characteristic of the incomplete real data. Table 3.4 reports the quantitative LDI generation results. OMLD again performs better than the baselines for the second layer components, which is the main focus of the works. Interestingly, PBO results are slightly superior on the first layer, since the OMLD formulation is more sensitive to ground truth noise, information holes and miss-detection (from Mask R-CNN). However, the results in the synthetic dataset encourage further improvement dependent on the quality of the available real datasets. We trained Tulsiani *et al*. with rendered source-target image pairs, as Stanford 2D-3D-S does not provide raw sequential trajectories appropriate for view synthesis, *i.e*. with high overlap between the views. To factor out the raw-rendered domain gap, as well as inconsistencies introduced by rendering artefacts in a pair of views, at test time we also use rendered images for both the source and the target image.

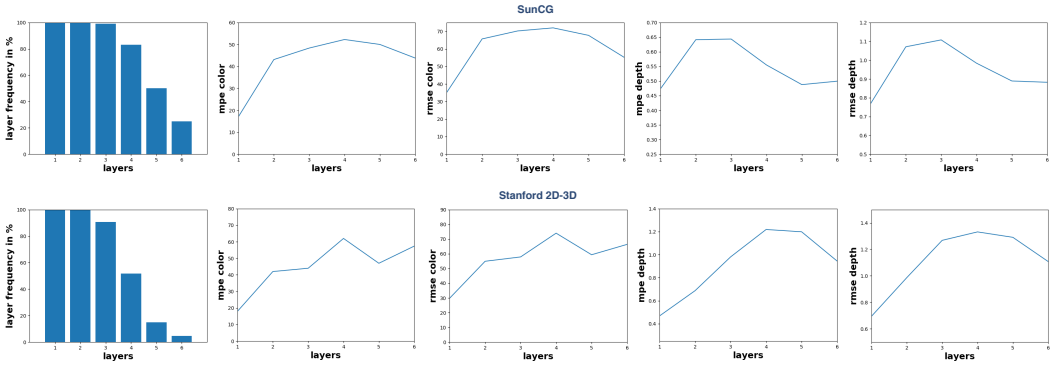


Figure 3.15. **Per-layer evaluation of OMLD on SunCG (top) and Stanford 2D-3D-S (bottom).** *Left:* Number of layer statistics per image. *Center:* MPE and RMSE errors for RGB. *Right:* MPE and RMSE errors for depth.

Figure 3.14 illustrates some qualitative results of OMLD on Standford 2D-3D-S. Note that the ground truth for the second layer is often sparse, despite the automatic selection procedure from Section 3.3.1. We only compute the evaluation in the subset of pixels for which ground truth is available.

Additionally we report the performance of all layers generated by OMLD. For every layer l, whenever there is no novel content (zeros), we migrate the information from the previous layer $l-1$. Then we compute the RMSE and MPE metrics between the predicted and the ground truth layers, only in the regions of interest, *i.e.* with novel content. The results are shown in Figure 3.15. The frequency plot on the left reports similar statistics for both datasets in the amount of layers per scene. We observe that most scenes require at least three layers. As expected, the first layer exhibits the lowest errors for texture and depth as it corresponds to visible, less ambiguous regions. The performance is comparable in the middle range of layers. Interestingly, we observe a performance increase in the last layers. We attribute this result to the increase of the contribution of the layout component in the composition of later layers. Regressing the layout, *i.e.* the empty box of the scene, is an easier problem than completing occluded object parts.

3.4.4 View synthesis

Synthesizing novel views is a direct application of the LDI generation task at hand. Therefore, we report view synthesis evaluations to compare the proposed methods against each other as well as state-of-the-art works. Starting with a source image I and an arbitrary transformation matrix T, the goal is to obtain a target image, which represents the content of I, after the source view has been multiplied by T. For this purpose, we leverage a rendering approach that warps the LDI layers in order, *i.e.* the information holes left by previous layers are filled by the following layers. Each layer's warping step is followed by morphological operations (dilation and erosion) to fill the small holes. The resulting texture images are further compared against the ground truth target images.

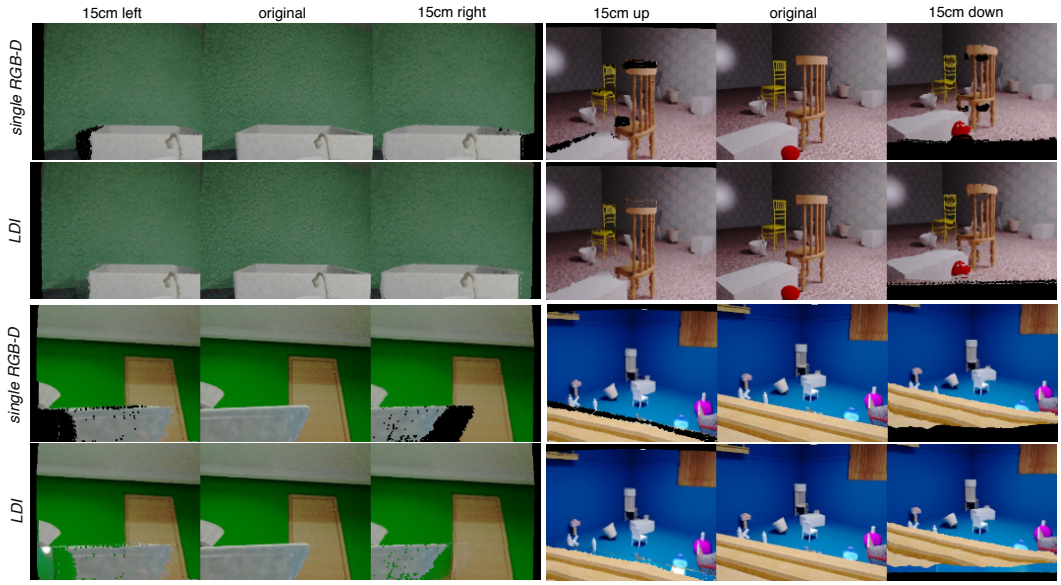


Figure 3.16. LDIs rendered on perturbed views on SceneNet. (*Upper row*): warping of a single RGB-D layer; (*Lower row*): warping of the proposed LDI. [©2019 Pattern Recognition Letters]

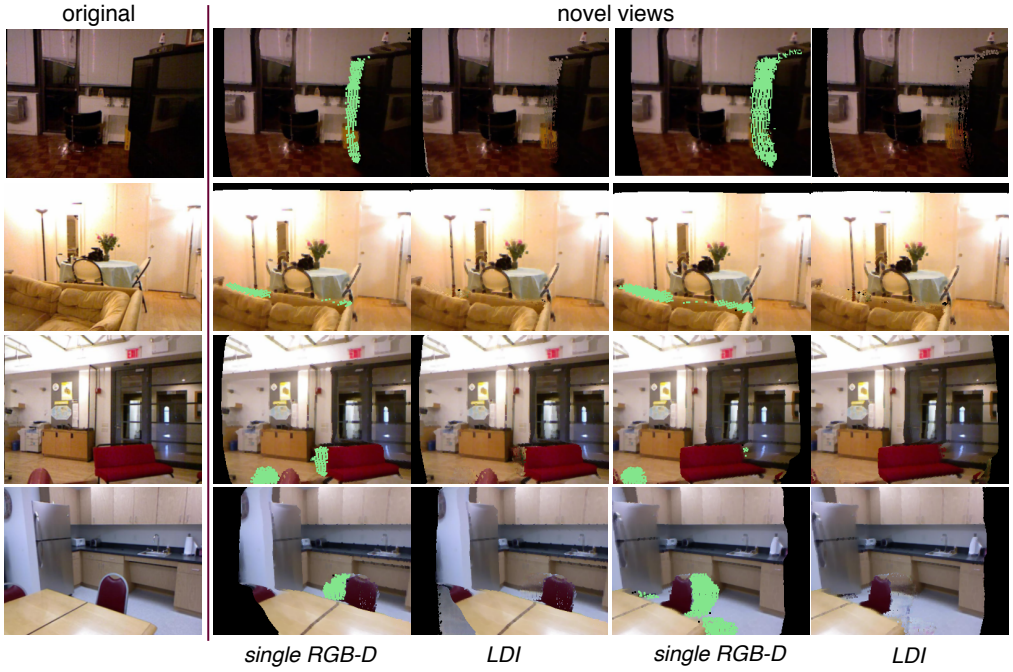


Figure 3.17. LDIs rendered on perturbed views, NYU v2. (*Left*): Input RGB. (*Right*): Novel views rendered from a single RGB-D vs. our LDI prediction. [©2019 Pattern Recognition Letters]

For SceneNet, we use a pair of consecutive frames (1 to 2 steps) from the original dataset as source and target images. To generate SunCG data pairs, we obtain target frames by perturbing the initial camera poses and rendering from the 3D mesh to the target view. In all view synthesis experiments, we utilize the same dataset splits as in the previous experiments.

Table 3.5. **View synthesis on SceneNet.** We evaluate the images on MPE, SSIM and PSNR, in range 0-255.

Method	SSIM ↑	MPE ↓	PSNR ↑
AF, [178]	0.53	51.00	17.83
AF, [178] (FCRN)	0.53	47.18	18.49
Tulsiani *et al.* [139]	0.58	46.41	18.14
PBO	**0.62**	**37.49**	**19.63**

Table 3.6. **View synthesis on SunCG.** The synthesized images are evaluated in terms of SSIM, MPE and RMSE, in range 0-255. [©2019 IEEE]

Method	SSIM ↑	MPE ↓	RMSE ↓
Tulsiani *et al.* [139]	0.33	71.36	87.09
PBO	0.56	29.01	49.89
OMLD	**0.65**	**18.19**	**34.71**

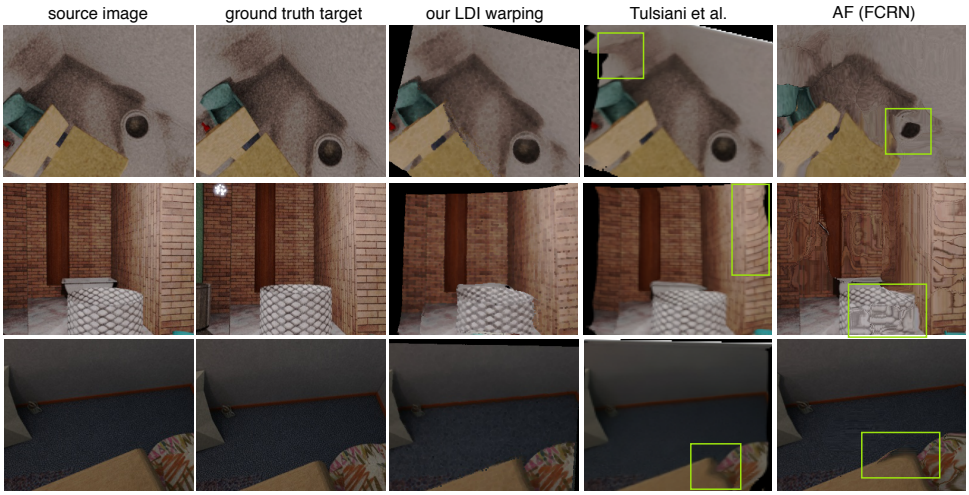


Figure 3.18. Qualitative comparison of PBO and baselines on novel view synthesis on SceneNet. [©2019 Pattern Recognition Letters]

To motivate the added value of a layered representation compared to a single-layer RGB and depth pair, we qualitatively evaluate view synthesis with random view perturbation, using the PBO model. In this experiment, the source view corresponds to the original input RGB image. We define the origin of the world coordinate system at the source camera. Thus the source pose has identity rotation $\mathbf{R}_{ref}$ and zero translation $\mathbf{t}_{ref}$. We modify either t_x or t_y in the reference translation vector $\mathbf{t}_{ref}$ to obtain a target pose with namely horizontal or vertical shift. Further, we utilize Eq. 3.1 to project the LDI layers, from source to target. The rendering results on SceneNet and NYU are shown namely in Figure 3.16 and 3.17. One can observe that, different from the single-layered model, our two-layered model populates the target view with additional pixels, that were originally occluded in the source image. The inpainting network can fairly complete the background, even when edges or relatively complex textures are present in the occluded background.

Next, we present comparisons of the proposed methods against state-of-the-art. At the time of publication of PBO [22] the only comparable methods were Appearance Flow (AF) [178] and [139], as they also receive a single input image and an arbitrary transformation. Other related view synthesis works explored multi-view inputs [31, 154, 176] or exploited scene geometry [32, 36] and their view generation is thus limited by the learned geometry. We trained their original models using code from their public repositories, on the same partitions of SceneNet as we trained PBO. The input and output are pairs of consecutive frames on

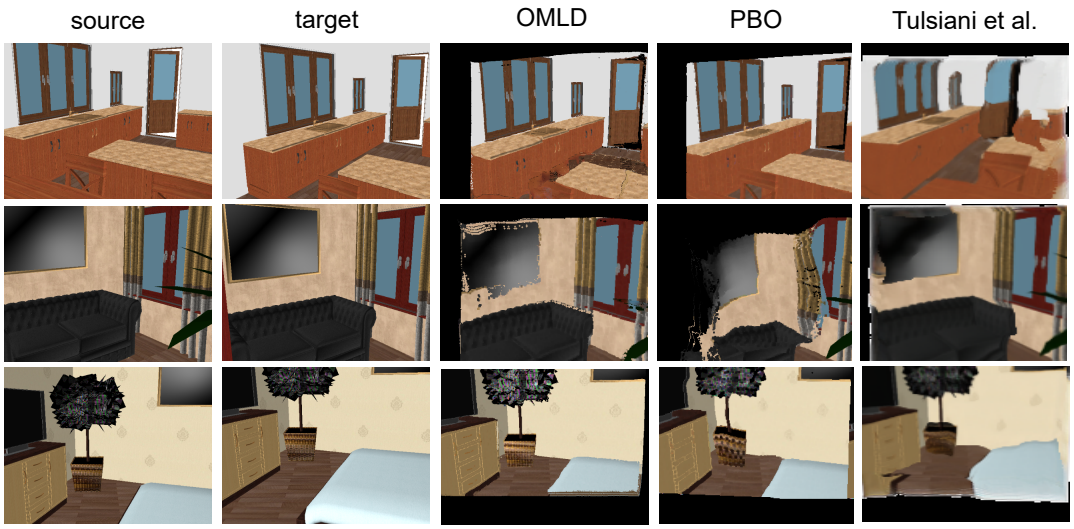


Figure 3.19. **View synthesis examples.** *Left:* Source image, *i.e.* the input to the proposed method as well as the target image, to be compared with the predictions. *Right:* Predicted novel views, using the LDIs from the proposed method, [22] and [139].[©2019 IEEE]

SceneNet. To factor out network capacity in our experiments, we also trained AF in another variant, based on the same model as in our depth-mask prediction branch[2]. Table 3.5 and Figure 3.18 report the comparison between the methods, and show that PBO outperforms [178] and [139] in all metrics, Mean Pixel Error (MPE), SSIM and PSNR. Moreover, the qualitative examples in Figure 3.18 confirm that our results preserve object shapes better and are more consistent with the ground truth target image.

We compared OMLD [21] against [139] and PBO [22]. We utilize the learned layered representation from each method and carry out the warping procedure to synthesize the target views. Table 3.6 reports the quantitative results. We observe a clearly better performance for OMLD in all metrics, as a consequence of more accurate color and depth layers. Figure 3.19 illustrates how OMLD is better at preserving the shapes of the objects during warping, and is better aligned with the target image.

3.4.5 Augmented diminished reality

Here we show another application of the investigated LDI generation methods, *augmented diminished reality*. Such manipulation of the scene content is not possible with other LDI inference works, as they do not rely on semantic or instance aware decomposition [45, 139].

Figure 3.20 illustrates the application of the proposed PBO method in diminished reality. First, we discard the foreground content to obtain the background layer. We then detect floor planes from the depth component in the background layer, such that other objects can be later added to the scene, in a geometry-aware manner. This is particularly interesting in interior design and furniture retail, to remove the current objects from an indoor space and experiment

[2] In their official repository, Zhou *et al.* [178] report improved performance when upgrading their architecture to fully convolutional networks.

Figure 3.20. The proposed PBO method, applied to augmented-diminished reality. (*Top*) Input RGB image (*Bottom*) Result after background inpainting and rendering of new furniture.

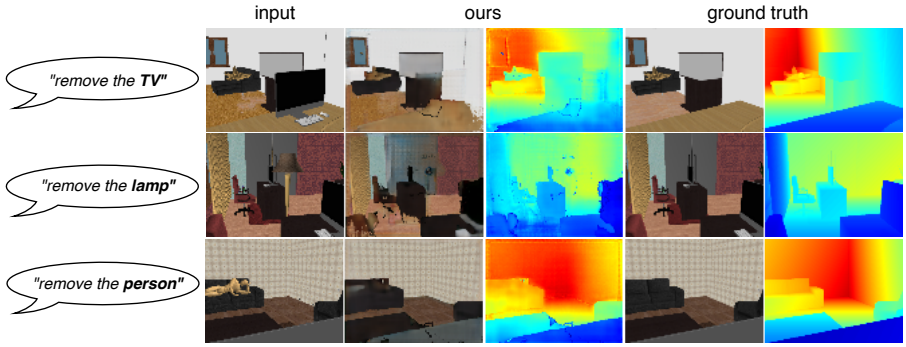


Figure 3.21. **Illustration of object removal results.** The category labels of the left indicate which object should be removed from the original image. We compare our predicted synthesized images (center) against the ground truth (right). [©2019 IEEE]

with different furniture configurations. We call such application *diminished-augmented reality* reconstruction from a single RGB image.

Object removal While the two-layer PBO model [22] enables the removal of the whole foreground, OMLD [21] allows for even more specific control at the instance level, *i.e.* removal of specific objects of interest from the image, as desired by the user. We take the generated layered representation before the sorting-based fusion, where each layer is either an object or the layout. Then, we specify the instance or object category to be removed from the input image. In the scope of this work, we assume fixed text descriptions of the form "*remove chair*". Nevertheless, employing natural language processing (NLP) approaches would be an interesting future exploration. Figure 3.21 illustrates examples of this novel task on SunCG. Interestingly, the generated images contain reasonable shapes for partially occluded objects, even when a good fraction of them is not visible.

3.5 Discussion and Current Trends

Since the rise of monocular LDI prediction task, pioneered by two orthogonal concurrent works of Tulsiani *et al*. [139] and PBO (ours), two main trends were established, which supervise the network either via view synthesis, or as a combination of depth reasoning and inpainting. The proceeding works that leverage similar layered representations also follow one of these trends for supervision [128, 137]. The most related work is 3DPI (3D Photo Inpainting) from Shih *et al*. [128]. The authors propose a layered depth inpainting approach that first uses an off-the-shelf depth prediction model, then utilizes depth discontinuities to define the inpainting areas. Finally an inpainting network is applied in these areas. The main difference from the OMLD formulation, is that the layer boundaries are not defined from semantic reasoning, but as depth boundaries. Comparing both methods, the main conceptual advantage of 3DPI is that it supports self-occlusion, since depth boundaries can locate within the same object. OMLD instead provides a better modeling for object shapes, due to the underlying semantic and instance reasoning, while enabling the additional image manipulation task. I believe that an interesting future exploration would be a combination of ideas from OMLD and 3DPI, which exploits semantics for more realistic object shapes, while leveraging the depth boundaries in each object layer to model self-occlusion.

3.6 Conclusions and Future Work

In this chapter we investigated two methods for decomposing a scene into a layered depth representation from a single RGB input. As a first proof-of-concept, we introduced PBO, a two-layer pipeline built over a feed-forward CNN for depth and mask prediction, and a GAN for background inpainting. We demonstrated how the additional information included in a layered depth map is a useful representation of the content of a scene with respect to standard depth prediction, particularly in dealing with occlusion in view synthesis. PBO outperformed the self-supervised [139] alternative by Tulsiani *et al*. in terms of layered representation and even view synthesis, despite the latter being trained via a view synthesis objective. In particular, we found that a model based on background inpainting is able to well explain the whole occluded surfaces, while a view synthesis model is bound by the perturbations it was exposed to during training (which results in less complete reconstructions). Importantly, the quality of the LDI prediction goes hand-in-hand with the accuracy of the individual components of our pipeline. As consequence, the performance of PBO gets naturally better as the fields of depth prediction and image inpainting advance.

To overcome the representational limitations of PBO, we additionally proposed an object-aware approach (OMLD), the first method to accommodate a flexible number of output layers, *i.e*. dependent on the scene complexity. We have demonstrated that the method outperforms previous works, especially on the invisible regions of partially occluded objects. Moreover, the evaluation proved that semantic awareness and the proposed re-composition mechanism improve the generation performance. Finally, the object-aware decomposition of

OMLD makes it possible to semantically manipulate the original input. We illustrated such capability by means of an object removal setting.

There is still a number of open challenges for the future. As our models rely on multiple components, *e.g.* mask prediction, depth inference, inpainting, failures in any component can lead to artefacts and affect the outcome. For instance, in case of repetitions in the detection for a certain object, OMLD would construct two layers rather than one.

More general, future research can be dedicated to modeling self-occlusion, making textures more crisp and realistic. For instance, one can represent an object instance with multiple values per pixel – instead of a single layer –, recognize the occlusion boundary within an object via depth discontinuities [128] and then inpaint the self-occluded object region.

4 Scene Graphs for Generation and Manipulation

Scene graphs refer to a data representation that describes physical scenes, where nodes and edges represent namely objects and inter-object relationships. This chapter of the dissertation explores scene graphs as a mean for scene synthesis and manipulation. Due to their modular and high-level nature, scene graphs are a convenient interface for controllable generation. Further, they are a low effort user interface for scene editing, which spares a few time consuming steps in the editing pipeline. Exemplary, instead of segmenting and inpainting a set of unwanted raw pixels from a photo, the user can simply remove the nodes from its graph for a similar outcome. Section 4.2 provides a formal definition of the utilized scene graph representation as well as the structure of the graph processing networks utilized in our methodology. In Section 4.3 we describe our method for semantic manipulation of images. We refer to this model as *SIMSG*, *i.e.* **S**emantic **I**mage **M**anipulation via **S**cene **G**raphs. We will show that SIMSG does not rely on direct supervision for image edits, *i.e.* no need for paired data of original and modified content. At the time of this research, there was no readily available real dataset that contains 3D scenes with scene graph annotations. In Section 4.4 we introduce 3DSSG, the semi-automatically obtained dataset of scene graphs associated with real 3D scene reconstructions [141], which we acquired to enable our works in 3D. Section 4.5 describes how we learn scene graphs from point clouds using the 3DSSG data. Finally, Section 4.6 presents Graph-to-3D, a model for 3D scene generation and manipulation based on the scene graphs from 3DSSG, which utilizes the predicted graphs from Section 4.5 as a means of cycle-consistency metric.

4.1 Related Work

4.1.1 Scene understanding

Scene understanding focuses not only on recognizing and localizing the objects present in the scene, but also their context and relationships. Here we visit the different representations in literature explored to parse a scene, with a focus on graph structures and their applications.

Scene graphs and images

Scene graphs is a term originally introduced in computer graphics where nodes are objects, and the edges represent relative transformations between the objects. The term has later been used in association with images, to provide abstract, structured representations of image

content. Isola *et al*. [57] utilize scene graphs to facilitate scene collaging, where the directed edges represent support relations between objects. Further, Johnson *et al*. [63] defined a scene graph as a directed graph structure that contains semantic labels for objects, their attributes and inter-object relationships, *i.e*. how they interact with each other. Following this high-level semantic representation, different methods have been proposed to infer scene graphs from an image input [46, 84, 85, 103, 112, 156, 158, 165]. Scene graph generation is usually formulated as detecting visual entities in the image (object detection) [117] and recognizing their interactions (visual relationship detection) [17, 59, 95, 121, 162]. In literature, scene graphs have been explored for a variety of tasks including image retrieval [63], conditional image generation [4, 65] and Visual Question Answering (VQA) [34]. We propose the first work that uses scene graphs for semantic image manipulation [19].

3D scene understanding: from object detection to scene graphs

An active research area within 3D scene understanding focuses on semantic segmentation [16, 26, 51, 109, 111] and object detection or classification [109, 110, 133, 175]. Holistic scene understanding [130] on the other hand predicts not only object semantics, but also the scene layout and sometimes even the camera pose [54]. Scenes are often represented through a hierarchical tree [83, 94, 127, 173], where the leaves are typically objects, which are grouped in scene components and functional entities in the intermediate nodes. A line of works use probabilistic grammar to parse scenes [94, 173]. Fisher *et al*. [30] exploit semantic scene graphs obtained via geometric rules on synthetic data, where relationships constitute spatial arrangements, enclosing or support.

Very recently the community started to explore semantic relationships on real world 3D environments. Armeni *et al*. [2] present a hierarchical mapping of 3D models of large spaces, organized in four levels: camera, object, room and building. The main difference of our 3DSSG dataset published in [141], is that we provide larger graphs with denser connections (see Table 4.3). Additionally, our focus is on more semantic inter-object relationships, which are difficult to obtain automatically, such as support. Moreover, as 3DSSG builds on a 3D dataset with changing scenes [140], it enables certain dynamic task, such as the proposed 3D scene retrieval in appendix A. and a potential for change detection and human activity analysis.

The aforementioned graph representations are utilized to explore tasks related to scene comparison [30], layout generation [96], object type predictions in query locations [179], as well as to improve 3D object detection [127] or 6D pose [75]. The research presented in this dissertation, additionally proposes the first work for scene graph prediction from a 3D scene (point cloud) [141] and introduces domain agnostic scene retrieval [141] as well as fully learned 3D scene generation [20].

4.1.2 Scene generation and manipulation

Images

Conditional image generation methods model the conditional distribution of images given some prior information, such as image labels [100, 105], attributes [157], lower resolution images [79], semantic segmentation maps [11, 147] and natural language descriptions [86, 116, 167, 170]. More general architectures enable translating from one image domain to another using paired [58] or unpaired training data [181]. Most relevant to our approach are methods that generate natural scenes from scene graphs [65] or layout [50, 172]. Johnson *et al*. use a graph convolution network (GCN) to translate the scene graph into a layout and further decode it into an image [65]. Our work published in [19] builds on this architecture while introducing additional mechanisms for information transfer from an input image, when the goal is image editing.

Image manipulation focuses on generating certain image parts while keeping the remaining image regions unchanged. A line of works performs automatic object-level image manipulation, such as editing of faces using attributes [13, 77, 171]. Other works propose image composition from a pair of foreground and background entities [5, 166]. At scene level the most common form of image manipulation is inpainting [108], which can be also conditioned on semantics [161] or user-specified contents [62, 174], as well as object removal [126]. A semantic map can also serve as editing interface for interactive image generation [147]. In our work [19] we instead propose using scene graphs as a lower-effort user interface, which allows not only change of the object labels, but also the inter-object relationships. Moreover, as we show later, a single general-purpose model supports a diverse set of editing scenarios. On another line, retrieval approaches such as Hu *et al*. [52] tackle image editing via a hand-crafted approach, which uses graphs to find a similar image patch from a database, suitable for replacement. In contrast, our framework allows for high-level semantic edits and can deal with object deformations which cannot be captured via copy-pasting. Yao *et al*. [160] explore 3D-aware manipulation of a scene by disentangling geometry from semantics. However, this approach is limited to a specific type of scenes (streets) and target objects (cars) and requires CAD models for training. Instead, our approach [19] tackles semantic changes of objects and relationships in natural scenes. Very recent related work focuses on interactive image generation from scene graphs [4] or layout [135]. These methods differ from ours in that they translate a graph or layout in multiple image variants, while our goal is to edit an *existing* image.

3D scenes

3D scene generation can be conditioned on various input modalities, including images, graphs and text descriptions. A line of works generates 3D scenes conditioned on images [104, 138]. Jiang *et al*. [60] propose scene synthesis, which is controlled via probabilistic grammar. Ma *et al*. [97] translate text to a scene graph – consisting of pairwise and group relationships – to then retrieve sub-scenes for 3D synthesis in a progressive manner. Graphs have been also applied to generate at the object level [101] with a hierarchical representation of parts. Very related are works that focus on scene layout generation. GRAINS [83] leverage graphs of hierarchical structure to synthesize 3D scenes, using a recursive VAE architecture that generates a scene layout, and uses retrieval from a database to populate the layouts

with object shapes. Luo *et al*. [96] incorporate a VAE based model to generate a 3D scene layout conditioned on a scene graph, combined with a rendering approach to obtain the final image output. Other relevant works use deep priors [144] or relational graphs [143] to learn object occupancy in the top-view of indoor environments. The auto-regressive architecture in [143] can additionally be used to edit a given scene, by adding new objects. Different from our work published in [20], these works either explore image-based representations as final output, or obtain 3D object shapes through retrieval in contrast to our end-to-end fully-learned scene generation. Moreover, while these models operate on synthetic scenes, we chose to work with real world datasets, to avoid influence of human-induced bias.

4.2 Scene Graph Formulation

Scene graph definition We define a semantic scene graph $G = (\mathcal{N}, \mathcal{R})$, as a set of object nodes $n_i \in \mathcal{N}$ and directed relationship edges $r_{ij} \in \mathcal{R}$ with $i \in \{1, ..., N\}$ and $j \in \{1, ..., N\}$, with N being the number of objects. In a vanilla scene graph, the nodes are represented as semantic object class categories, *i.e.* $n_i = c_i$. Throughout this work, we additionally utilize augmented scene graph representations, where nodes are extended with other modalities, such as bounding box parameters b_i, *e.g.* $n_i = (c_i, b_i)$. In all cases, the edges are comprised of semantic predicate categories. Each relationship is characterized by an outbound object (*subject*), the edge label (*predicate*) and the inbound object (*object*), *i.e.* a triplet *subject - predicate - object*.

Graph processing network In our graph processing networks we require a mechanism that allows information to flow between objects, along their connecting relationships. As scenes come with diverse complexities, we want the graph processing models to allow flexibility in the number of input nodes. Thus, we employ a Graph Convolutional Network (GCN) similar to [65], to process the acquired triples. Each layer l_g of the GCN operates on directed relationships triplets (subject – predicate – object) which we denote (*out – p – in*) and consists of three steps. First, each triplet ij is fed in a Multi-Layer Perceptron (MLP) $g_1(\cdot)$ for message propagation

$$(\psi^{(l_g)}_{out,ij}, \phi^{(l_g+1)}_{p,ij}, \psi^{(l_g)}_{in,ij}) = g_1(\phi^{(l_g)}_{out,ij}, \phi^{(l_g)}_{p,ij}, \phi^{(l_g)}_{in,ij}). \tag{4.1}$$

Second, the aggregation step combines the information coming from all the edges of each node

$$\rho^{(l_g)}_i = \frac{1}{M_i}\Big(\sum_{j \in \mathcal{R}_{out}} \psi^{(l_g)}_{out,ij} + \sum_{j \in \mathcal{R}_{in}} \psi^{(l_g)}_{in,ji}\Big) \tag{4.2}$$

where M_i is the number of edges for node i, and $\mathcal{R}_{out}$, $\mathcal{R}_{in}$ are the set of edges of the node as out(in)-bound objects. The resulting feature is linearly projected through a final update MLP $g_2(\cdot)$

$$\phi^{(l_g+1)}_i = g_2(\rho^{(l_g)}_i). \tag{4.3}$$

Alternatively, inspired by [82], we adapt a residual graph connection to overcome potential Laplacian smoothing on graphs. Throughout this chapter, I will refer to this node update variant as *residual GCN*.

$$\phi_i^{(l_g+1)} = \phi_i^{(l_g)} + g_2(\rho_i^{(l_g)}). \tag{4.4}$$

The final features $\phi_{out,ij}^{(l_g+1)}, \phi_{p,ij}^{(l_g+1)}, \phi_{in,ij}^{(l_g+1)}$ are then processed by the next convolutional layer l_g, in the same fashion. After each layer l_g, the node visibility is propagated to a further neighbour level. Hence, the number of layers equals the order of relations that the model can capture.

4.3 Semantic Image Manipulation (SIMSG)

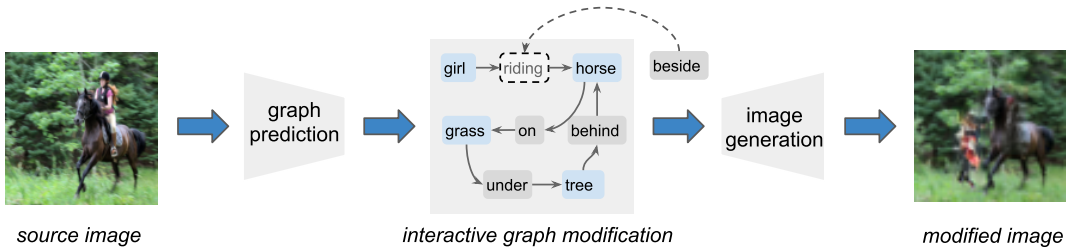


Figure 4.1. **Semantic Image Manipulation at inference time.** Given an image, we infer a semantic scene graph. The user makes changes in the graph's nodes and edges. Then, our image generation network gives an edited version of the input image, which respects the constellations in the modified graph.[©2020 IEEE]

The focus of this work is to perform semantic manipulation of images using a scene graph as interface for requesting the changes. Given an image I as input, we infer its scene graph $\mathcal{G}$ which will serve as interaction interface with a user. Further, we generate a new image I' from the edited scene graph $\tilde{\mathcal{G}}$ in combination with the original content of I. An overview of the method is shown in Figure 4.1. At inference time, our method consists of three core components. First, we encode the image contents in a spatio-semantic scene graph, designed so that it can easily be manipulated by a user. Second, the user modifies the given scene graph at the *node-level* to change object categories and locations in the image, or at the *edge-level* to change the semantic relationships between objects. Finally, the manipulated image is generated from the modified graph.

A practical limitation in this task is the difficulty in obtaining training data with pairs of source and target images annotated with scene graphs. To overcome this limitation, we demonstrate a method that does not rely on direct supervision for image edits, and instead learns through an unsupervised image reconstruction proxy task. Thus, the image manipulation task is learned in a self-supervised way. Note that the underlying system components, *e.g.* graph generation, are still learned with full supervision, utilizing image-graph pairs.

At the core of our method lies a training mechanism that randomly masks patches from the image as well as information from the scene graph, and then aims to reconstruct the same image. We explain this technique in Section 4.3.2. The full training pipeline can be seen in Figure 4.2. First, a graph generation network is used to extract information from the input image, such as bounding boxes and a semantic scene graph (Section 4.3.1). Then, the scene graph is processed (Section 4.3.3) and arranged in a 2D layout (Section 4.3.3). Further, the generated layout and input image are fed in a decoder to synthesize the final image (Section 4.3.3).

4.3.1 Graph generation

The task of scene graph generation from an image consists in describing an input image with a semantic graph $\mathcal{G} = (\mathcal{N}, \mathcal{R})$, of objects $\mathcal{N}$ (nodes) and their relationships $\mathcal{R}$ (directed edges). Since this is a well-researched problem [84, 103, 156, 165], in our work we integrate a state-of-

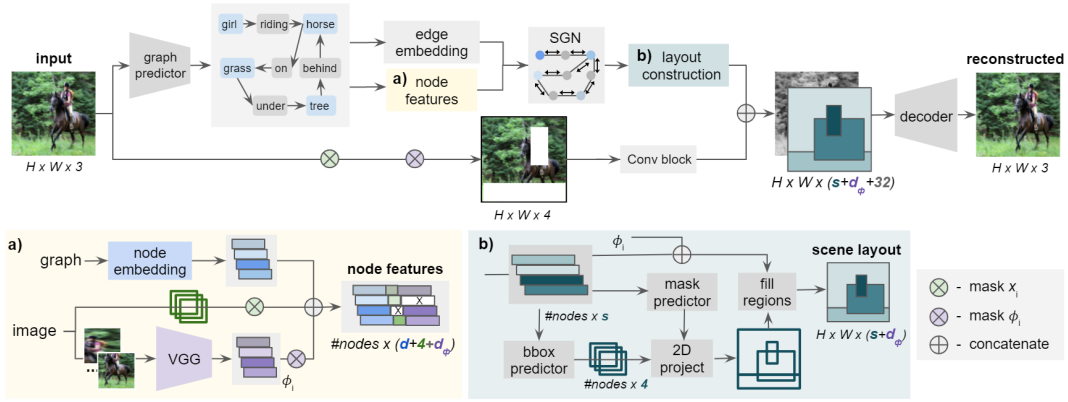


Figure 4.2. **Overview of the SIMSG supervision.** *Top:* Given an input image, we predict the respective scene graph and use it to reconstruct the input from a corrupted representation. a) The graph nodes n_i *(blue)* encompass class embeddings, bounding box coordinates x_i *(green)* and visual features ϕ_i *(violet)* from cropped objects. We randomly mask boxes x_i, visual features ϕ_i and patches in the input image. The model then learns to reconstruct the same graph and image utilizing the persisting information. b) The per-node feature vectors are projected to image space to form a layout, using the predictions from the SGN.[©2020 IEEE]

the-art method for scene graph prediction (F-Net) [84], trained independently from the other components. The method clusters the scene graph into subgraphs of objects and their (partial) relationships, to reduce computation. Controllable image generation would profit from more detailed descriptions than the semantic node and edge categories in the vanilla scene graph. Therefore, we propose a graph representation that constitutes an augmentation of the vanilla scene graph with visual (neural) features and spatial locations. We thus define object nodes as triplets $n_i = (c_i, \phi_i, x_i) \in \mathcal{N}$, where $c_i \in \mathbb{R}^d$ is a d-dimensional, learned embedding of the object category and $x_i \in \mathbb{R}^4$ represents the four values defining the bounding box of the object (top, left, bottom, right). $\phi_i \in \mathbb{R}^{d_\Phi}$ is a visual feature encoding of the object which can be obtained by feeding the object image patch to a convolutional neural network (CNN), here VGG-16 [129], pre-trained for image classification (ImageNet [18]). Similarly, for a given relationship describing an object pair (i, j), the method learns a class embedding ρ_{ij} of the relationship class $r_{ij} \in \mathcal{R}$.

Importantly, our augmented scene graph contains sufficient information to represent the appearance and location of objects while allowing for control in the preservation of each component. For instance, in a relationship change scenario, one can preserve appearance features of an object, while the corresponding locations and relationships are changed. In a object replacement scenario instead, we are interested in preserving the location (bounding box coordinates) and discarding the initial appearance features.

4.3.2 Training mechanism

Training the model with full supervision would require annotations of the form $(I, \mathcal{G}, \mathcal{G}', I')$ where a pair of related images I and I' that exhibit a change is annotated with scene graphs, namely $\mathcal{G}$ and $\mathcal{G}'$. Given the difficulty in acquiring ground truth $(I', \mathcal{G}')$, our goal is to

train a model supervised only by $(I, \mathcal{G})$ through a reconstruction proxy task. Consequently, we generate annotation quadruplets $(\tilde{I}, \tilde{\mathcal{G}}, \mathcal{G}, I)$ using the available data $(I, \mathcal{G})$ as the *target* supervision and synthesize $(\tilde{I}, \tilde{\mathcal{G}})$ through a random masking procedure that operates on the node level. This idea is inspired by common image inpainting works, which create input pairs $\tilde{I}$ and I, where $\tilde{I}$ is a corrupted version of the image, containing missing patches of information. During training, the object neural features ϕ_i are masked with an arbitrary probability p_ϕ. Independently, we mask the bounding box coordinates x_i with probability p_x. In practice, the information of the masked node is replaced by zeros. Additionally, when occluding graph information, image regions corresponding to the hidden nodes are also occluded. Effectively, this masking mechanism transforms the editing task into a self-reconstruction, or a denoising problem. The network will learn to complete the missing parts in the corrupted graph and image, utilizing the remaining information.

At inference time, once the graph nodes or edges are edited by a user, the image regions subject to modification are occluded, and the network, having learned to reconstruct the image from the scene graph, will create a reasonable modified image. In the example of Figure 4.1, the user wishes to apply a change in the way the `girl` interacts with the `horse`, editing the predicate label from `riding` to `beside`. Since we expect the spatial arrangement to change, our system occludes the bounding boxes x_i, localizing these objects in the original image. Then, their new extents and positions $\hat{x}_i$ will be estimated considering the layout of the rest of the scene (*e.g.* grass, trees). To encourage this change, the system should automatically mask the original image regions related to the target objects. Simultaneously, to ensure that the *visual identities* of `horse` and `girl` are preserved through the change, their visual feature encodings ϕ_i must remain unchanged.

4.3.3 Graph to image model

Spatio-semantic scene graph network

To process our augmented graph representation we leverage a spatio-semantic scene graph network (SGN). The SGN network learns a graph transformation that allows information to flow between objects, along their relationships. Given a graph $\tilde{\mathcal{G}}$ with partially occluded information, the task of the SGN is to learn meaningful neural representations per object that will be then used to reconstruct the image. This is done via a series of convolutional operations on the graph structure.

After T graph convolutional layers, the last layer predicts one latent representation per object (node) $\psi_i \in \mathbb{R}^s$. This output object representation is further processed through two separate MLPs, in order to predict bounding box coordinates $\hat{x}_i \in \mathbb{R}^4$, and a spatial binary mask $\hat{m}_i \in \mathbb{R}^{M \times M}$ indicating pixel-wise object extent. Predicting coordinates for each object is a form of reconstruction, since object locations are known and are already encoded in the input n_i. As we show later, this is needed when modifying the graph, for example for a new node to be added. As described in the following section, the predicted object representation will be then reassembled into the spatial configuration of an image, referred to as the scene layout.

Scene layout

This component is responsible for transforming the graph-structured representations predicted by the SGN back into a 2D spatial arrangement of features, which can then be decoded into an image. To this end, we use the predicted bounding box coordinates $\hat{x}_i$ to project the masks $\hat{m}_i$ in the proper region of a 2D feature map of the same resolution as the input image. We concatenate the original visual feature ϕ_i with the node features ψ_i to obtain a final node feature. The projected mask region is then filled with the respective features, while the remaining area is padded with zeros. This process is repeated for all objects, resulting in $|\mathcal{N}|$ tensors of dimensions $(d_\phi + s) \times H \times W$, which are aggregated through summation into a single layout for the image. The output of the layout component is an intermediate representation of the scene, which contains enough information to reconstruct an image.

Image synthesis

The last part of the pipeline is a decoder architecture, which synthesizes a target image, combining the information in the source image I and the generated layout. We explore two different decoder architectures, cascaded refinement networks (CRN) [11] (similar to [65]), as well as SPADE [107], originally proposed for image synthesis from dense semantic segmentation. Different from the original works [11, 65, 107], we condition the image synthesis on the source image by concatenating the predicted layout with extracted low-level global features from the source image. Note that prior to image feature extraction, regions of I are occluded using the mechanism explained in Section 4.3.2. We fill the occluded regions with Gaussian noise to encourage diversity for the generator.

4.3.4 Loss objective

We train the image reconstruction model via a combination of loss terms, common in image synthesis. The bounding box prediction is trained by minimizing the L_1-norm: $\mathcal{L}_b = \|x_i - \hat{x}_i\|_1^1$, with weighting term λ_b. Due to unavailability of binary masks in the given datasets, the mask prediction is instead learned in a self-supervised way, via attention. The image synthesis task is supervised by adversarial training, using two discriminators, similar to [65]. First, a global discriminator D_{global} encourages consistency over the entire image. A local discriminator D_{obj} in turn operates on each reconstructed object region to ensure that the generated patches look realistic. We additionally apply an auxiliary classifier loss [105] to ensure that the appearance of the generated objects fairly represents their real labels. Finally, we apply a reconstruction loss term $\mathcal{L}_r = \|I - I'\|_1$ to enforce image content preservation in regions that have not been changed. The total image synthesis loss is then

$$\begin{aligned}\mathcal{L}_{\text{synthesis}} = \mathcal{L}_r + \lambda_g \min_G \max_D \mathcal{L}_{\text{GAN,global}} \\ + \lambda_o \min_G \max_D \mathcal{L}_{\text{GAN,obj}} + \lambda_a \mathcal{L}_{\text{aux,obj}},\end{aligned} \tag{4.5}$$

where λ_g, λ_o, λ_a are loss weighting factors and

$$\mathcal{L}_{\text{GAN}} = \mathop{\mathbb{E}}_{q \sim p_{\text{real}}} \log D(q) + \mathop{\mathbb{E}}_{q \sim p_{\text{fake}}} \log(1 - D(q)), \tag{4.6}$$

where p_{real} is the ground truth distribution (of either an object or the image) and p_{fake} corresponds to the distribution of fake objects or images, while q is the input to the discriminator which is sampled from p_{real} or p_{fake}. In our SPADE version, we additionally employ a perceptual loss term based on VGG-19 $\lambda_p \mathcal{L}_p$ and a GAN feature loss term $\lambda_f \mathcal{L}_f$ following the original implementation [107]. Moreover, following SPADE, D_{global} becomes a multi-scale discriminator.

4.3.5 Implementation details

Graph prediction network To acquire scene graphs for the experiments on Visual Genome (VG) we utilized a state-of-the-art scene graph prediction network [84], which we trained using the official code repository [1] provided by the authors. Thereby, we use the default parameters from the original paper, with a batch size of 8 images for 30 epochs. Following the commonly used pre-processing from [17], we obtain a subset of Visual Genome (sVG) with 399 object and 24 predicate classes. To avoid overlap between the training and test data for the image manipulation model we utilize the dataset splits from [65].

SGN architecture details. The SGN network is composed of 5 graph convolutional layers. The propagation and update units are essentially 2-layer MLPs, where the hidden units have a size of 512 and the output units a size of 128. The last layer of the SGN returns the node features $s = 128$, binary masks (16×16) and bounding box coordinates via a 2-layer MLP with a hidden unit dimension of 128. During training, the visual features ϕ_i and bounding box coordinates x_i are randomly masked with independent probabilities of $p_\phi = 0.25$ and $p_x = 0.35$ respectively. The class embedding vectors that represent the object c_i and predicate labels r_i have a size of 128 each. Each node n_i is a concatenation of c_i with four bounding box coordinates x_i (top, left, bottom, right) and the visual features ϕ_i ($d_\phi = 128$) in the image patch within the box. To extract the neural features we leverage a pre-trained VGG-16 architecture [129] followed by a linear layer.

CRN architecture details. The CRN architecture consists of 5 cascaded refinement modules [11], with namely 1024, 512, 256, 128 and 64 output channels. Each module consists of two 3×3 convolutions, each followed by batch normalization [56] and Leaky-ReLU. The output of each module is concatenated with the down-sampled initial CRN input, before being fed to the next layer. The input to the first layer is the concatenation of the generated layout and the masked image global features. We report results for images with a 64×64 resolution. The object discriminator is only applied on the image regions that have been changed.

SPADE architecture details. We employ a SPADE architecture with 5 residual blocks [107]. Each block results in a tensor with namely 1024, 512, 256, 128 and 64 channels. In particular, the layout is fed in the SPADE normalization layer, to modulate the layer activations, while the image global feature is concatenated with the modulation result. We use a global discriminator D_{global} with two scales. Also here, the object discriminator is only applied on the changed image regions.

[1] https://github.com/yikang-li/FactorizableNet

Global feature extraction branch details. The (randomly) masked image regions are populated with standard Gaussian noise. Image features are extracted using a convolutional layer that results in 32 channels, with kernel size 1×1, followed by batch normalization and ReLU activation. Additionally, a binary mask is concatenated with the image features that indicates the regions of interest (noise), so that the network can easily identify these regions.

Training settings. We trained the graph-to-image model with Adam optimization [70] with an initial learning rate of 10^{-4}. The batch size for the 64×64 images is 32, while for 128×128 is 8. All objects in an image batch are fed as a single batch to SGN, visual feature extractor and discriminator. The weighting factors for the loss terms are namely $\lambda_g = 0.01, \lambda_o = 0.01, \lambda_a = 0.1, \lambda_b = 10$ for CRN and $\lambda_g = 1, \lambda_o = 0.1, \lambda_a = 0.1, \lambda_b = 50, \lambda_f = 10, \lambda_p = 10$ for SPADE. Training on an Nvidia RTX GPU takes about 3 days for Visual Genome and 4 hours for CLEVR, for images of size 64×64. All models on VG were trained for 300k iterations, while CLEVR models are trained for 40k iterations.

4.3.6 Evaluation

We evaluate our SIMSG method quantitatively and qualitatively on common image reconstruction and generation metrics. To measure image reconstruction, we report results for standard metrics such as the structural similarity index (SSIM), the mean absolute error (MAE) as well as perceptual error (LPIPS) [168]. To assess the image generation quality and diversity, we report the common inception score (IS) [122] and the FID [47] metric. In absence of a real scene graph dataset with source-target pairs, we rely on synthetic data for a full quantitative evaluation on image edits, to complement our evaluation. Therefore we perform experiments on two datasets, CLEVR [66] and Visual Genome [73]. As CLEVR is a synthetic dataset, obtaining automatic ground truth pairs for image editing is feasible, which allows quantitative evaluation of SIMSG. Experiments on Visual Genome (VG) instead illustrate our method's capabilities in a real, much less constrained and more diverse setting. As image editing cannot be quantitatively evaluated in VG, we evaluate an image inpainting proxy task and compare against a conditional version of sg2im [65].

Manipulation types

Our model supports a variety of modification types at inference time, depending on the user interaction with the graph. These manipulation modes include object removal, addition and replacement, as well as relationship changes. Here we define how each editing mode is practically carried out. Note that it is technically possible to perform more complex manipulations as a sequence of these elementary changes.

Object removal: A node is entirely removed from the scene graph together with all its connecting inbound and outbound edges. The source image region corresponding to the node area is masked out (occluded).

Object replacement: The semantic label of a node is assigned to a different category. The visual encoding ϕ_i of that node is masked out (set to zero), as it describes the old object which

we want to replace. The location component of respective bounding box x_i is preserved, to keep the novel object in place, while the size component is replaced by the bounding box estimated from the SGN, to fit the new semantic category. Note that in case the user wants to specify an object appearance from a query image, ϕ_i is replaced by the neural feature of the query object instead of being masked out.

Relationship change: This operation changes the semantic label of an edge ij. The goal is to preserve the subject i and object j visual features as much as possible but change their interaction, *e.g.* `<sitting>` to `<standing>`, often leading to re-positioning of objects or a pose change. Thus, the respective neural features ϕ_i and ϕ_j are maintained, while the bounding box coordinates x_i and x_j are masked (set to zeros). Both the original and the newly generated image regions are occluded, to enable inpainting of the dis-occluded region and new object generation.

Object addition: A new node is added to the graph, in which the semantic label is set as desired by the user. As there is no information about the location or appearance of the new object, x_i and ϕ_i are set to zeros. The respective image region defined by the predicted $\hat{x}_i$ is also occluded. The user can additionally specify connecting edges between the newly added node and the existing objects.

Baselines

We adapt the sg2im model [65] to an image manipulation baseline. We dub this model *Conditional sg2im baseline (Cond-sg2im)*. Since the original sg2im method synthesizes images from a scene graph input only, without a source image, their image generation network consists of the layout concatenated with normal noise. Instead, we condition this network on the input image by replacing the noise component with this image. Similar to SIMSG, we mask image areas corresponding to the objects being reconstructed, before the concatenation. All loss terms are the same as for SIMSG. Note that this baseline supports all manipulation types, except from relationship change. Additionally, we employ a *fully-supervised baseline (full-sup)* – with the same architecture as Cond-sg2im – which assumes source and target pairs of images and the corresponding graphs. The entire source image and target scene graph are provided as input to the model. The model is trained by optimizing the $\mathcal{L}_1$ objective to the ground truth target image. The other loss terms remain the same as in our model. Additionally, we utilize ISG [4] as baseline, which was state-of-the-art in interactive scene generation from a scene graph at the time of publication [19]. Since we are mainly interested in semantically rich and diverse relationships, we trained [4] on Visual Genome, utilizing their official code repository. Note that originally ISG was trained on COCO with automatically obtained scene graphs. Since Visual Genome lacks segmentation masks, we disable the mask discriminator from ISG.

Synthetic data

We leverage the CLEVR framework from [66] to generate a dataset of image and scene graph editing pairs $(I, \mathcal{G}, \mathcal{G}', I')$. We obtain 21,310 pairs of images (128×128), split into 80% for training, 10% for validation and 10% for testing. The data pairs contain the same scene under

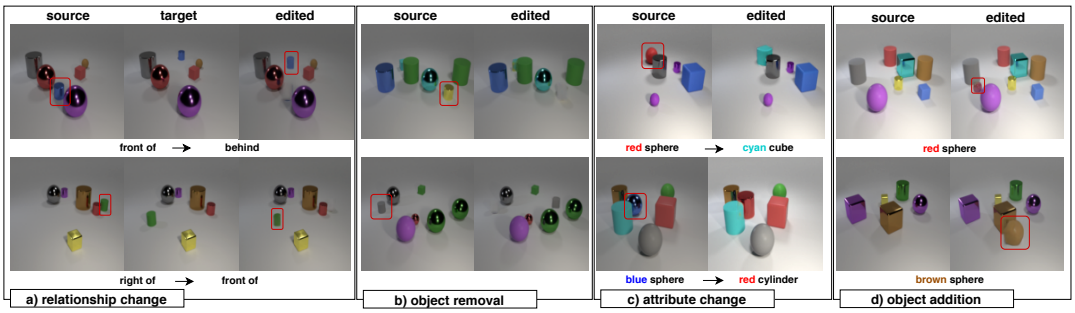


Figure 4.3. **Image manipulation on CLEVR** We illustrate different scene manipulation types, including relationship change between two objects, object removal and object replacement (corresponding to attribute changing).[©2020 IEEE]

Method	All pixels				RoI only	
	MAE ↓	SSIM ↑	LPIPS ↓	FID ↓	MAE ↓	SSIM ↑
Full-sup	6.75	**97.07**	0.035	**3.35**	9.34	93.49
SIMSG (CRN)	7.83	96.16	0.036	6.32	10.09	93.54
SIMSG (SPADE)	**5.47**	96.51	0.035	4.73	**7.22**	**94.98**

Table 4.1. **Image manipulation on CLEVR.** We compare our method against a fully-supervised baseline. [©2020 IEEE]

a change, such as addition, removal, positional change or attribute change. The images contain 3 to 7 randomly shaped and colored objects. We obtain predicates from the relative positions of objects in different object pairs – {`front of`, `behind of`, `left of`, `right of`}. Additionally, the dataset includes annotations of bounding boxes. This enables the evaluation of our method with exact ground truth for the manipulation task. We train our model as described, *i.e.* without making use of the image pairs, and compare our approach to the fully-supervised baseline.

In Table 4.1 we report the mean SSIM, MAE, LPIPS and FID on CLEVR for the manipulation task (including object replacement, removal, addition and relationship change). Note that IS would not be appropriate for CLEVR, as it would compare the generated synthetic images against a real image distribution. Our SIMSG method performs either better or comparable to the fully-supervised baseline on the reconstruction metrics, which shows the capability of generating meaningful changes, *i.e.* close to the ground truth. Though many outputs are valid, this is still an indication of a correct outcome, especially for the learning of object attributes. The FID results are instead better for the fully-supervised setting. This suggests that additional supervision for pairs, if available, would potentially lead to improvement in the visual quality, *e.g.* less artifacts. Figure 4.3 illustrates the qualitative performance of SIMSG on CLEVR in four different modes: relationship changes (a), object removal (b), object addition (c) or replacement (changing its attributes) (d). To facilitate visualization, we highlight the modified area with a bounding box.

Figure 4.4. **Visual feature encoding.** Comparison between the baseline (top) and our method (center) on Visual Genome for object reconstruction. The scene graph remains unchanged; an object in the image is occluded, while ϕ_i and x_i are kept. Our neural features ϕ_i preserve appearance characteristics when the objects are masked out from the source image. [©2020 IEEE]

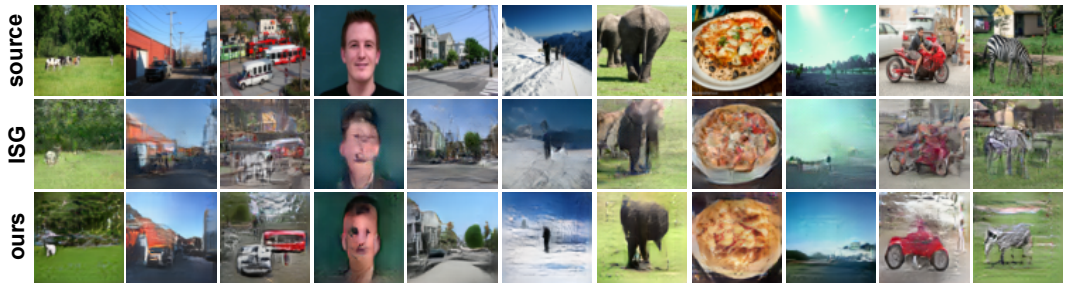


Figure 4.5. **Qualitative results comparing SIMSG with CRN decoder and ISG [4]** in the fully-generative setting. The entire image is masked and reconstructed using the per-object visual feature information.

Real images

We evaluate SIMSG on Visual Genome (VG) [73] to assess its performance on real images. We use the VG v1.4 dataset with the splits as proposed in [65]. After the pre-processing step of [65] the dataset features 178 object categories and 45 relationship types, where each split consists of 62,565 train, 5,506 validation, and 5,088 test images with scene graph annotations. We evaluate our models with ground truth (GT) scene graphs on all the images of the test set. For the experiments with predicted scene graphs (P), an image filtering takes place (*e.g.* when no objects are detected), therefore the evaluation in performed in 3874 images from the test set. We observed edge duplicates in the ground truth scene graphs which can lead to ambiguity for some manipulation tasks. For instance, when we change only one of the duplicate edges, the object pair can contain two conflicting relationships (*e.g.* `on` and `beside`). Hence, we remove such repetitions once one of the duplicate edges is edited.

In absence of ground truth pairs for manipulations, we only report quantitative evaluation on image reconstruction. In this case, objects (one at a time) are masked out in the original image and we evaluate the reconstruction task. This task can be interpreted as semantically conditioned inpainting. Nevertheless, we illustrate the method on the manipulation setting qualitatively, as no ground truth pairs are neccessary.

Method	Decoder	All pixels					RoI only	
		MAE ↓	SSIM ↑	LPIPS ↓	FID ↓	IS ↑	MAE ↓	SSIM ↑
ISG [4] (Generative, GT)	Pix2pixHD	46.44	28.10	0.32	58.73	$6.64_{\pm 0.07}$	-	-
SIMSG (Generative, GT)	CRN	41.57	33.9	0.34	89.55	$6.03_{\pm 0.17}$	-	-
SIMSG (Generative, GT)	SPADE	41.88	34.89	0.27	44.27	$7.86_{\pm 0.49}$	-	-
Cond-sg2im [65] (GT)	CRN	14.25	84.42	0.081	13.40	$11.14_{\pm 0.80}$	29.05	52.51
SIMSG (GT) w/o ϕ_i	CRN	9.83	86.52	0.073	10.62	$11.45_{\pm 0.61}$	27.16	52.01
SIMSG (GT) w/ ϕ_i	CRN	**7.43**	**88.29**	0.058	11.03	$11.22_{\pm 0.52}$	**20.37**	**60.03**
SIMSG (GT) w/o ϕ_i	SPADE	10.36	86.67	0.069	8.09	$12.05_{\pm 0.80}$	27.10	54.38
SIMSG (GT) w/ ϕ_i	SPADE	8.53	87.57	**0.051**	**7.54**	$\mathbf{12.07}_{\pm 0.97}$	21.56	58.60
SIMSG (P) w/o ϕ_i	CRN	9.24	87.01	0.075	18.09	$10.67_{\pm 0.43}$	29.08	48.62
SIMSG (P) w/ ϕ_i	CRN	7.62	88.31	0.063	19.49	$10.18_{\pm 0.27}$	22.89	55.07
SIMSG (P) w/o ϕ_i	SPADE	13.16	84.61	0.083	16.12	$10.45_{\pm 0.15}$	32.24	47.25
SIMSG (P) w/ ϕ_i	SPADE	13.82	83.98	0.077	16.69	$10.61_{\pm 0.37}$	28.82	49.34

Table 4.2. **Image reconstruction on Visual Genome.** We report the results for SIMSG (ours) and the baselines, using ground truth scene graphs (GT) and predicted scene graphs (P). (Generative) refers to results in a fully generative setting, *i.e.* the whole input image is occluded. [©2020 IEEE]

Reconstruction task. Here we want to evaluate our networks' reconstruction capabilities in two different settings, fully and partially generative. The results are reported in Table 4.2. In the *fully generative* setting (Generative), we occlude the entire input image and utilize the per-object encoded features ϕ_i and bounding boxes x_i for reconstruction. We compare our model against ISG [4], who also use per-object features and a scene graph as input. Table 4.2 (Generative) reports comparable reconstruction errors, while SIMSG clearly dominates when a source image is provided as input (*c.f.*SIMSG (GT)). This confirms our choice of directly manipulating an existing image, rather than simply fusing different node features. Unsurprisingly, inception score (IS) and FID mostly relate to the decoder network, where SPADE outperforms CRN and Pix2pixHD. For the *partially generative* setting, we are interested in reconstructing a single object in the image, while the rest is kept unchanged. The evaluation is performed over all objects in the test set. In particular we want to ablate the visual feature ϕ_i and quantify its influence in visual appearance encoding. We use all the associated object bonding boxes x_i, class labels and neural features (w/ ϕ_i) to condition the SGN. However, the region of the image corresponding to the reconstructed object is occluded. Table 4.2 shows the reconstruction error a) over all pixels and b) in the object area only (RoI). We report the same learned model without (w/o ϕ_i) visual features. Here ϕ_i is set to zeros. We test SIMSG in two different settings; using ground truth graphs (GT) and graphs predicted from the input images (P). As expected, activating the visual features of the missing region leads to improvement of the reconstruction metrics (MAE, SSIM, LPIPS). On the other hand, the IS and FID metrics are not considerably affected, as they do not measure pair-wise similarity between corresponding ground truth and synthesized images. Table 4.2 shows that while the CRN and SPADE decoders perform similarly in reconstruction metrics (CRN is slightly better), SPADE dominates for the FID and inception score, suggesting higher generation quality.

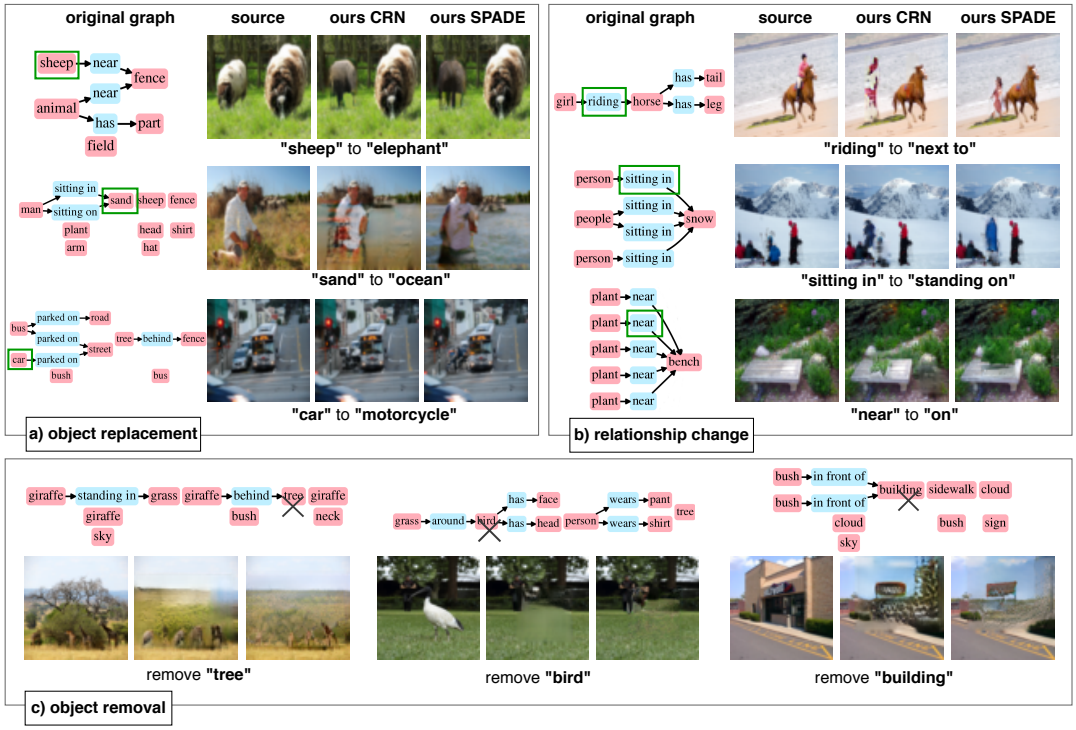


Figure 4.6. **Image manipulation** Given the source image and the ground truth scene graph, we edit the image by changing the semantic labels of the graph. We illustrate **a)** object replacement, **b)** relationship changes, and **c)** object removal. Green bounding box highlights the changed node or edge.[©2020 IEEE]

Figures 4.4 and 4.5 illustrate visually the image reconstruction qualitatively, namely in the partial and full generation setting. Figure 4.4 shows that both SIMSG and the cond-sg2im baseline, generate reasonable object shapes in accordance with their semantic category. However, as our SGN is aware of the objects' visual features, appearance characteristics from the original image are successfully preserved. In practice, this capability is particularly beneficial during a relationship change, *i.e.* when the objects of an image are re-positioned, while their identity is preserved. In Figure 4.5 we observe similar results as ISG, even though our method is not specifically trained for the fully-generative task.

Main task: image editing. Here we test our method's potential in making manipulations in an image as a result of interactively editing the scene graph, *i.e.* changing nodes or edges in the graph. Figure 4.6 illustrates qualitative results in three manipulation modes —object removal, object replacement and relationship changes. The method is capable of diverse replacements (a), ranging from objects to background instances. Notably, the novel entity adapts to the image context, *e.g.* the ocean on the second row does not occlude the person, which would be expected in standard image inpainting with a bounding box as mask. A more challenging and interesting scenario is to change the relationship between objects, which typically involves re-positioning. Figure 4.6 (b) shows that the model understands different semantic relations, such as `sitting` vs. `standing` and `riding` vs. `next to`. The objects are re-positioned in a reasonable way, according to the semantic relationship change. For object removal (c), one can observe that the method performs well at inpainting regions with uniform texture, but can also handle more complex textures (left example).

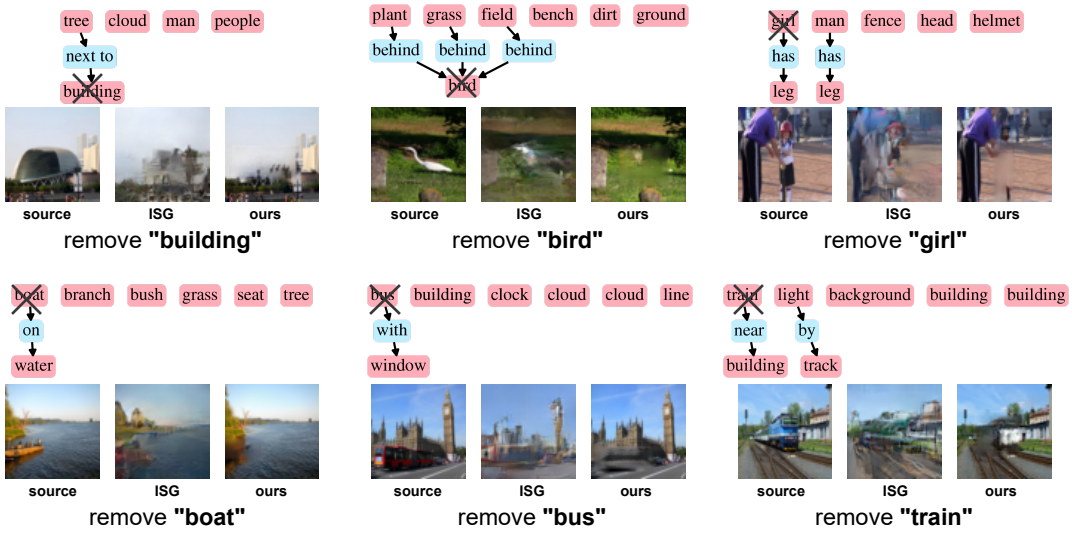


Figure 4.7. Qualitative results comparing SIMSG with CRN decoder against ISG on object removal.

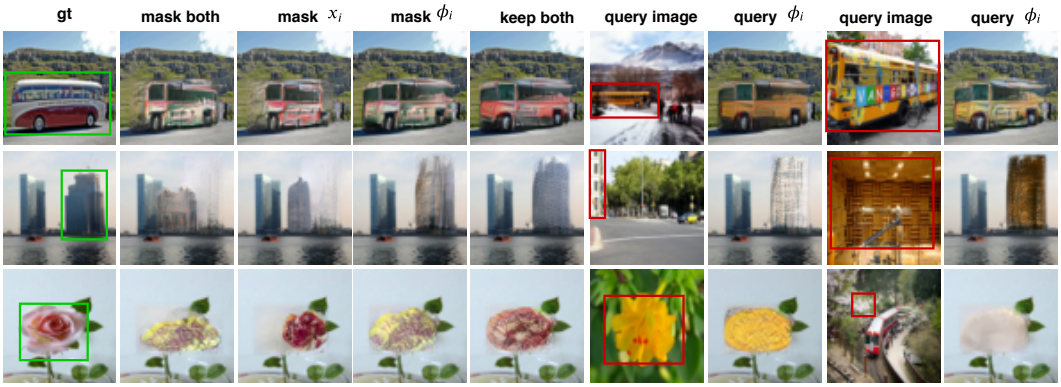


Figure 4.8. **Ablation of the node components** We illustrate the effect of the proposed node components, in all their possible combinations - *i.e.* with vs. without bounding boxes x_i and visual neural features ϕ_i. In the case of using a query image, we extract visual features of an object (indicated with a red box) and update the node of an object of the same category in the input image.[©2020 IEEE]

Interestingly, when we remove the building (example on the right), the originally hanging sign is placed in the bush. Additionally, we adapt [4] for object removal by removing a node and its connecting edges from the input graph (same as in ours), while the visual features of the remaining nodes (coming from our source image) are used to reconstruct the rest of the image. Figure 4.7 demonstrates that our method performs generally better, since it is intended for direct manipulation on an image.

Component ablation. Figure 4.8 qualitatively ablates the node components in the proposed augmented scene graph. For a given image, we occlude a region corresponding to a certain node which we want to reconstruct. We experiment with all possible combinations of masking and keeping the bounding box coordinates x_i and neural features ϕ_i from the scene graph representation. One task of interest would be to inpaint the missing region with another object of the same category (*e.g.* changing color or style). Thus we additionally explore a setting in which external neural features ϕ are extracted from a query object in another

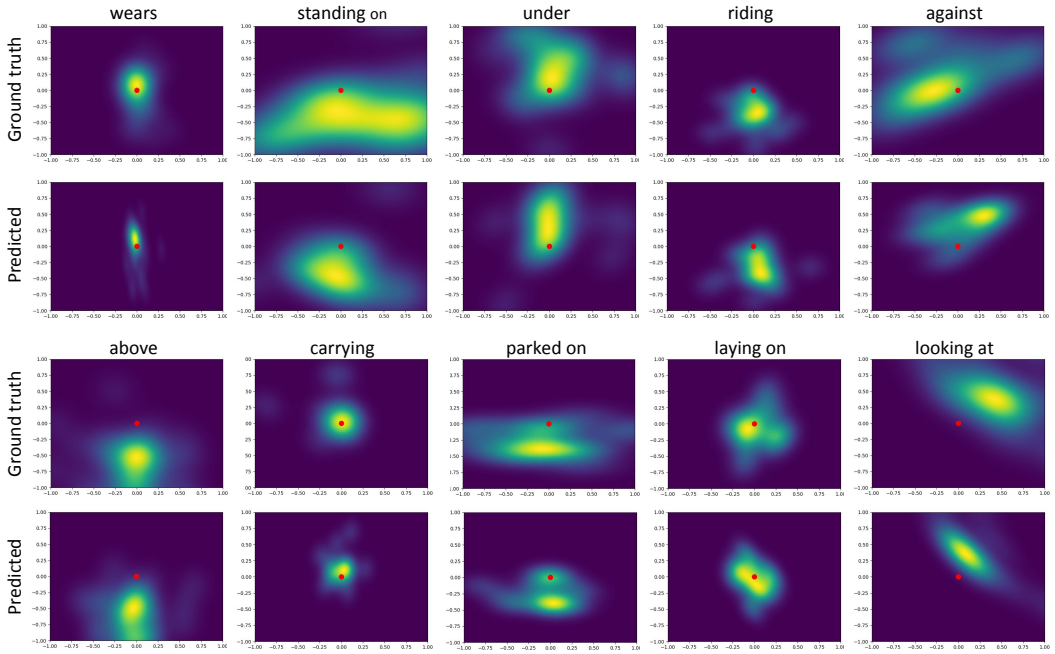


Figure 4.9. **Heatmaps generated from object and subject relative positions for a subset of predicate categories.** The object is centered at $(0, 0)$ and the relative position of the subject is calculated. The heatmaps are generated from the relative distances of centers of the object and subject of a pair. *Top:* Ground truth boxes. *Bottom:* our inferred boxes after masking the location information from the scene graph.

image. As expected, masking the box coordinates leads to a change in the size and location of the newly generated object, while masking the visual features ϕ results in changed object appearance.

Spatial distribution of predicates Another aspect of the performance that we want to assess is, how good our model has learned to accurately localize new objects in relation to objects already existing in the scene. As box prediction is a one-to-many problem, a metric that directly compares the predicted and ground truth box would be ill-posed. Therefore, we instead visualize the heatmaps of relative placement between box pairs. In particular, for every triplet (*i.e.* subject - predicate - object) we predict the subject and object bounding box coordinates $\hat{x}_i$, by masking the respective ground truth boxes from the graph. From there, for every triplet we extract the relative object-subject distance between the box centers, which are later grouped by predicate category. Figure 4.9 visualizes the heatmaps of the ground truth and predicted bounding box distributions per predicate. We observe similar distributions, in particular for relationships that are spatially meaningful, such as `wears`, `above`, or `riding`.

Failure cases Despite the encouraging results in many diverse scenarios and manipulation types, we identify a few failure cases of the SIMSG method. First, in this task we want to prevent the encoder from "copying" the whole RoI, which is not desired for instance if we want to change the relationships of a deformable instance, *e.g.* from `sitting` to `standing`. As a side effect, while the model is able to retain general appearance information, some visual properties of changed objects are sometimes not recovered. For instance, the color of the

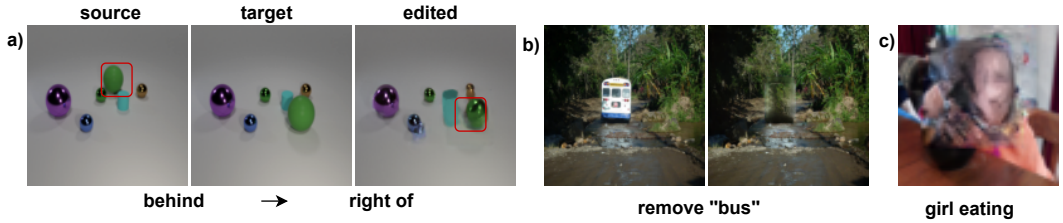


Figure 4.10. Illustration of failure cases of our model, related to a) partial feature encoding, b) not modeled dependence between nodes and c) underrepresented/difficult scenarios.

green object in Figure 4.10 a) is preserved while the material is wrongly altered. Second, the model does by default not adapt regions corresponding to unchanged nodes as a consequence of a related change. For instance, shadows or reflections do not follow the re-positioned objects, if those are not nodes of the graph and explicitly marked as changing subject by the user, Figure 4.10 b). In addition, similarly to other methods evaluated on Visual Genome, the quality of some close objects remains limited, *e.g.* close-up of people eating, Figure 4.10 c). Also, we have observed that often having a node `face` connected to animals, typically results in a human face. This can be attributed to the fact that, most face annotations in the Visual Genome dataset are related to a human. Last, we found it challenging to generate images that preserve the graph constrains in higher resolution. For instance, our model with SPADE decoder is capable of generating plausible images in 128×128 resolution, whereas the relationship changes are often not met.

4.4 Dataset with 3D Semantic Scene Graphs

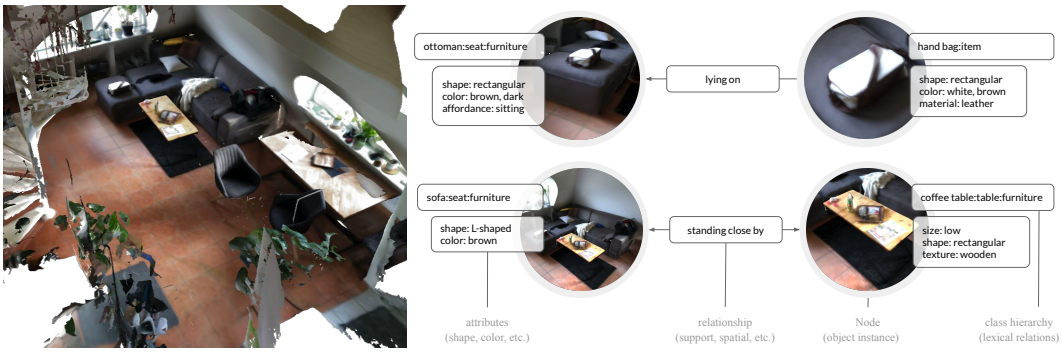


Figure 4.11. **3DSSG Scene graph representation** including hierarchical class labels c and attributes A per node, as well as relationship triplets between the nodes.[©2020 IEEE]

With the goal of generating and manipulating 3D scenes via a semantic graph, we want to collect a large-scale dataset suitable for these tasks. Most importantly we wish to have sets of real 3D scenes and the corresponding semantic graphs. We are particularly interested in working with real environments, to capture real-world patterns in object arrangements, and reduce human induced bias. Thereby, we introduce 3DSSG which provides 3D semantic scene graphs for 3RScan [140], a large scale, real-world dataset with around 1.4k 3D reconstructions of 478 changing indoor environments. An exemplary semantic scene graph $\mathcal{G} = (\mathcal{N}, \mathcal{R})$ in 3DSSG is illustrated in Figure 4.11. Notably, the nodes represent 3D object instances in a 3D scan, which in contrast to previous works [2, 16, 73, 140], are not assigned to a single object category, but instead are defined by a class hierarchy $c = (c_1, ..., c_d)$ where $c \in \mathcal{C}^d$, and d can vary. In addition to these object categories each node contains a list of attributes A that describe the visual and physical properties of the object. Thereby, one interesting type of attributes are affordances [153], which specify possible functions of an instance, such as *sitting* or *eating*. Since we deal with dynamic environments, we draw a connection between affordances and the object states. The edges in the 3DSSG graphs define semantic relationships (predicates) between the nodes such as `lying on`, `standing in`, `higher than`, `same as`. To obtain the labels in 3DSSG we combine human annotations with geometric data and additional manual verification to ensure good quality graphs. In summary, 3DSSG contains 1482 scene graphs with 48k object nodes and 544k edges. Interestingly, 3D scene graphs can easily be *rendered* to 2D. Given a 3D scene and a camera pose, it is possible to extract the graph part that is present in that image. Note that support and attribute comparison relationships do not depend on the reference view and therefore remain unchanged, while directional relationships (`behind`, `front`, `left`, `right`) need to be recomputed for the novel (camera) viewpoint. Considering the 363k raw RGB-D images with camera poses of 3RScan, this results in 363k 2D scene graphs. Table 4.3 provides a comparison of our 3DSSG dataset with the only available real 3D scene graph dataset from Armeni *et al*. [2]. More information and statistics about 3DSSG are provided in appendix C. In the following sections I will describe the different entities of our 3D semantic scene graphs.

dataset	size	instances	classes	obj. rel.
Armeni *et al*. [2]	35 buildings 727 rooms	3k	28	4
3DSSG (Ours)	1482 scans 478 scenes	48k	534	40

Table 4.3. Comparison between 3D scene graph datasets. [©2020 IEEE]

4.4.1 Nodes

The nodes in our graph represent object instances in a 3D scene. Each instance is defined by a class hierarchy c where c_1 is the corresponding annotated label from 3RScan. The subsequent labels (c_{i+1}) are obtained as a result of recursive parsing of the lexical definition of the previous step c_i via WordNet [28]. For instance, the definition *"chair with a support on each side for arms"* gives us c_{i+1} = `chair` as a hypernym for c_i = `armchair`. Note that due to lexical ambiguities, this algorithm results in multiple interpretations of a class label, *e.g*. an academic chair vs. a sitting chair. Therefore, we perform a manual selection step, to obtain a single definition per class label that is most likely referring to an object in an indoor environment. Thus the dataset provides 534 lexical descriptions and the respective class hierarchies, one per each class label. Figure 4.12 visualizes the lexical hierarchy on a small partition of classes. The complete lexical tree containing all the labels can be found in appendix C.

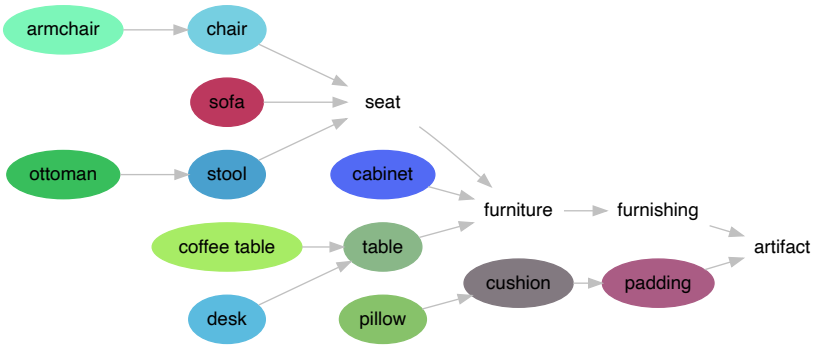


Figure 4.12. Lexical hierarchical sub-tree on a small subset of object class labels. The extended version can be found in appendix C. [©2020 IEEE]

4.4.2 Attributes

Attributes are semantic labels which augment the graph nodes with explicit object properties. In the following we define the different types of attributes and describe their acquisition. Given the semantic diversity and the large number of instances in the dataset, we put particular attention in the efficiency of label extraction and annotation.

Static Properties describe visual object features including color, shape, size, texture and physical properties such as (non-)rigidity. For size related attributes, geometric data is used in

combination with class labels to identify the relative size of the objects within the same class category. Since some features are class specific, we exploit lexical descriptions to assign them automatically on the class level, *e.g.* a ball is `spherical`. More complex attributes cannot be automatically annotated, since they are instance specific. This includes, for example, material (`metal, ceramic`), shape (`round, squared`) or texture (color or pattern). We design an interface to let expert annotators manually label them. Note that since the 3RScan dataset contains multiple scans of the same scene, many object instances repeat among the different scans. Since static attributes do not change, we annotate them in the reference scan and further replicate to each rescan.

Dynamic Properties refer to attributes, which describe properties that are subject of change in dynamic scenes. In the 3DSSG dataset we refer to them as *states*, including `open / closed`, `on / off` or `full / empty`. State categories are inherently class specific, *e.g.* doors can be open or closed, while a couch can not. Their current condition, however, is particular to an instance and time of scanning. Therefore, state annotation is also carried out manually, together with the static properties.

Affordances We define affordances as object functionalities or interaction possibilities of nodes of a specific object class *e.g.* a `plate` is for `eating`.This definition follows previous works [2, 35, 153]. Differently, since we work with changing scenes, we condition them on their dynamic state attribute. For instance, only a `opened door` can be `closed`. These changes are often induced by human activity (see examples in appendix C). Overall, 3DSSG contains 93 different attributes on approximately 21k object instances which amount to a total of 48k attributes.

4.4.3 Relationships

We define three main categories in 3DSSG , which will be described in the following paragraphs: a) support relationships, b) spatial or proximity relationships and c) comparative relationships.

Support Relationships Support relationships denote the supporting structures of the entities in a scene [102]. In real 3D scans, automatically extracting support relationships is quite challenging in practice, due to data noise and partiality. For instance, information holes in a scan, do not always guarantee a contact point between the child and parent structure. Thus our algorithm first finds potential support candidates, followed by verification steps. For each object, we extract the neighbouring instances which are present within a radius (*e.g.* 5cm) as potential support candidates. These support candidates then go through a manual verification step to eliminate wrong assignments and complete missing relationships. The remaining class support pairs are then annotated with a more specific semantic label (*e.g.* `lying on, leaning against`) and then specified for each instance in the dataset. Note that an instance can have one, multiple or no supports. For instance, walls are by default supported by the floor, a shelf can be supported by two different walls, and the floor is the only instance that does not have any support.

Proximity Relationships Proximity or spatial relationships indicate the relative position between two objects (*e.g.* `behind of`, `next to`, `left of`). As some of them have a directional nature (`left`) one needs to establish a reference view to fully define the relationship description. We compute proximity relationships automatically, based on geometric rules, between the nodes that share a support structure. A bag on a table therefore has no proximity relationship with a chair. Note that this proximity could be automatically derived from the relations of its support parent (table) with the chair. Thus this choice reduces the annotation redundancy between instances.

Comparative Relationships As the name suggests, comparative relationships are derived via a comparison between object attributes, *e.g.* `smaller than`, `brighter than`, `cleaner than`, `same color as`. We obtain these attributes automatically, leveraging the aforementioned attributes from Section 4.4.2.

4.5 Scene Graph Prediction from a Point Cloud

In this section we introduce the Scene Graph Prediction Network (SGPN) that learns from 3DSSG data. Starting from a scene s given as a point set $\mathcal{P}$, as well as the class-agnostic instance segmentation $\mathcal{M}$, the objective is to generate a scene graph $\mathcal{G} = (\mathcal{N}, \mathcal{R})$, describing the object class categories (nodes) in the scene $\mathcal{N}$ as well as the class categories of their relationships (edges) $\mathcal{R}$, Figure 4.13.

Note

Recently in literature instance segmentation can also assume class labels (alongside of instance labels). To make clear that we only rely on instance labels, here we use the term *class-agnostic* instance segmentation. This work focuses on the estimation of node and class labels, therefore we utilize such instance information directly from the 3RScan reconstructions. In theory, every 3D geometric segmentation method that is able to segment separate instances could be used to generate this input.

4.5.1 Architecture

Following works in scene graph prediction from images [95, 156, 158], we want to extract visual features per node ϕ_n and edge ϕ_r. Thereby, we employ two PointNet [109] networks to obtain ϕ_n and ϕ_r, which we name ObjPointNet and RelPointNet respectively. Thus, for a scene s, we extract the point set of each individual instance i, masked with $\mathcal{M}$

$$\mathcal{P}_i = \{\delta_{m_k i} \odot p_k | k = \{1, ..., |\mathcal{P}|\}\} \tag{4.7}$$

where p, m are instances of $\mathcal{P}, \mathcal{M}$, δ denotes the Kronecker delta [2], and $|\cdot|$ is the number of points in $\mathcal{P}$. Having obtained the instance-level point sets $\mathcal{P}_i$, we feed them separately to ObjPointNet .

Further, to extract neural features that represent an edge, we obtain a point set for every node pair (i,j). Thereby, we leverage the union of the two 3D bounding boxes $\mathcal{B}$ of the pair, and extract all points that lie within this union box

$$\mathcal{P}_{ij} = \{p_k | p_k \in (\mathcal{B}^i \cup \mathcal{B}^j), k = \{1, ..., |\mathcal{P}|\}\}. \tag{4.8}$$

The input to RelPointNet is thus a point set representing an edge $\mathcal{P}_{ij}$, concatenated with a helper mask $\mathcal{M}_{ij}$, that indicates the instance correspondence in $\mathcal{P}_{ij}$. The mask has the value one for points belonging to instance i, two if the point belongs to instance j and zero otherwise. As preserving the reference view of the edge $\mathcal{P}_{ij}$ is important to infer directed proximity relationships like `left` or `right`, we do not apply rotational augmentation in the data. The scale of both the object and edge point sets is also left unchanged as size can

[2] $\delta_{ij} = 1 \iff i = j$

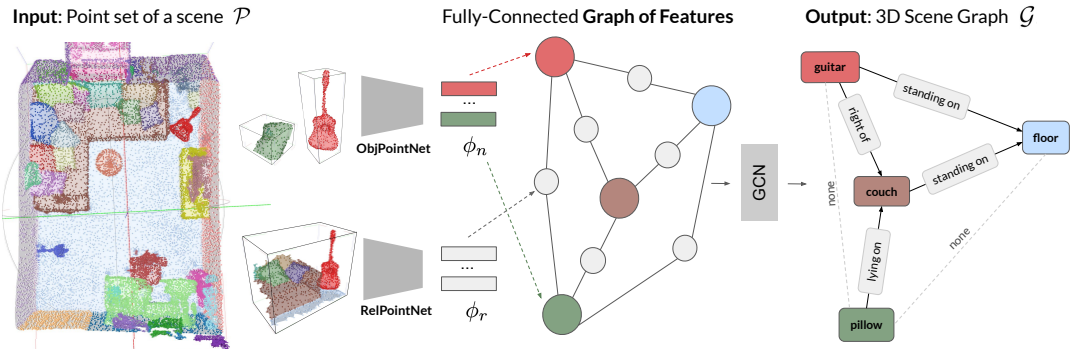


Figure 4.13. **Scene Graph Prediction Network** Starting with a point set $\mathcal{P}$ of a scene, and its class-agnostic instance segmentation $\mathcal{M}$, we estimate a scene graph $\mathcal{G}$. *Left:* Neural point features ϕ are extracted for each instance and edge. *Center:* The features ϕ are organized in a graph form for further processing from a GCN. *Right:* The resulting graph consists of semantic labels for object nodes and edges.[©2020 IEEE]

provide meaningful information to infer object categories. We, however, normalize the center of the object and edge point clouds to zero.

Once all features are extracted, we arrange them in a graph structure, such that we can process triples of the form (`subject, predicate, object`). Thereby, ϕ_n occupy subject and object units, while edge features ϕ_r occupy the predicate units. We employ a residual GCN to process the relationship triplets. At the last GCN layer, we leverage two MLPs for the prediction of the final node and predicate class categories.

4.5.2 Loss objective

We supervise SGPN with a joint object classification loss $\mathcal{L}_{\text{obj}}$ and predicate classification loss $\mathcal{L}_{\text{pred}}$ which are simply added together as

$$\mathcal{L}_{\text{total}} = \lambda_{\text{obj}} \mathcal{L}_{\text{obj}} + \mathcal{L}_{\text{pred}} \tag{4.9}$$

where λ_{obj} is a loss weighting factor. Based on observations on the natural world, we direct the attention to the fact that for a certain object pair there are multiple valid relations that describe their interaction. Exemplary, a chair can be `close by` another chair, and simultaneously have the same size (`same size as`). With this motivation, instead of the standard multi-class cross entropy, we formulate the predicate loss $\mathcal{L}_{\text{pred}}$ as per-class binary cross entropy. This way, edge labels are inferred independently. For both loss terms we employ focal loss as it is more robust with respect to class imbalance [87]

$$\mathcal{L} = -\alpha_t (1 - p_t)^\gamma \log p_t \tag{4.10}$$

where p_t represents the prediction logits and γ is a hyper-parameter. α_t is the normalized inverse frequency in the case of multi-class loss ($\mathcal{L}_{obj}$) and a fixed factor (indicating edge / no-edge) for the per-class predicate loss ($\mathcal{L}_{pred}$).

4.5.3 Implementation details

We adopt two standard PointNet architectures for node and edge feature extraction. The input points to ObjPointNet have three channels to accommodate a 3D point, while the RelPointNet inputs have four (one extra channel for the helper mask). The size of the resulting features ϕ_n and ϕ_r is 256. The GCN is implemented with $l = 5$ residual layers, where $g_1(\cdot)$ and $g_2(\cdot)$ (see Section 4.2) are composed of a linear layer followed by a ReLU activation. The MLPs for class prediction consist of three linear layers each, with batch normalization and ReLU activation. We utilize $\lambda_{obj} = 0.1$. For the per-class binary classification loss, α_t is set to 0.25. We use an Adam optimizer, where the learning rate is 10^{-4} and the scene batch size is 1.

4.5.4 Evaluation

Method	Relationship		Object Class		Predicate	
	R@50	R@100	R@5	R@10	R@3	R@5
① RelPred Baseline	0.39	0.45	0.66	0.77	0.62	0.88
Single Predicate, ObjCls from PointNet	**0.46**	0.52	**0.69**	**0.79**	0.70	0.85
② Multi Predicate, ObjCls from PointNet	0.41	**0.67**	**0.68**	**0.78**	**0.92**	**0.96**
Multi Predicate, ObjCls from GCN	0.30	0.60	0.60	0.73	0.79	0.91

Table 4.4. Evaluation of the scene graph prediction task on 3DSSG. We present triples prediction, object classification as well as predicate prediction accuracy. [©2020 IEEE]

In the following, we report results of our 3D graph prediction evaluated on our newly created 3DSSG dataset, utilizing the same train and test splits as originally proposed by 3RScan [140]. Since our scene graphs are quite dense and diverse in labels (see statistics in appendix C) a pre-processing of the ground truth graph data was necessary to train a learned model. We split the original graphs into subgraphs of size 4–9 nodes based on their 3D location. Furthermore, we only consider a subset of the object and relationship classes and discard the most underrepresented categories. After the pre-processing we effectively use 160 different classes for objects and 26 for predicates. For reproducibility purposes we have made these splits publicly available.

Baselines

In absence of related works that predict scene graphs from 3D data, we compare the method against a relationship prediction baseline, inspired by [95]. We re-implemented and adapted their method to work with 3D data. The baseline extracts node and edge features from an image, which we translate to PointNet features in 3D, similar to our network. The edge and node features are passed directly, namely to a predicate and object classifier, each having

three fully connected layers followed by batch norm and ReLU. We validate the effectiveness of our multi predicate classifier and GCN in our proposed network in an ablation study.

Metrics

Following previous works [95, 156] we base our evaluation on a top-k recall metric (R@k). The top-k metric checks if the ground truth label falls in the first k prediction scores, sorted in decreasing order. We first evaluate predicate and object classification separately. Additionally, we evaluate the relationship triplet prediction. Since SGPN predicts the object categories independently from the predicates, there are no natural triplet classification scores as output of the network. Thus we obtain an ordered list of triplet classification scores by multiplying the respective subject, object and predicate scores of a certain triplet [158]. Note that for fairness of comparison, in the multiple predicate prediction variant, we consider `none` as the most dominant (top-1) score if for all predicate categories the binary prediction score is smaller than 0.5.

Results

Table 4.4 reports the quantitative evaluation. We outperform the baseline in graph related metrics, such as predicate and triplet prediction, while being comparable in object classification. As expected, the multiple predicate prediction model leads to a better performance for predicates, which we relate to the ambiguity in a single-answer classification problem, when multiple outputs are reasonable. Moreover, we report two versions of the model, in which the object classification branch is applied a) directly on the PointNet features ϕ_n and b) to the output node features of the GCN. We observe a slight dominance for the former in terms of object and predicate prediction accuracy. Figure 4.14 illustrates examples of the resulting scene graphs. In all nodes and edges we show the predicted labels together with the respective ground truth in brackets. We observe that most node and edge label predictions are reasonable. Often, misclassifications are justifiable, *e.g.* predicting desk instead of table, second row. Interestingly, predicate false positives often make sense, *e.g.* third row: orange trashcan is on the right of the green trash can, even though ground truth was not available.

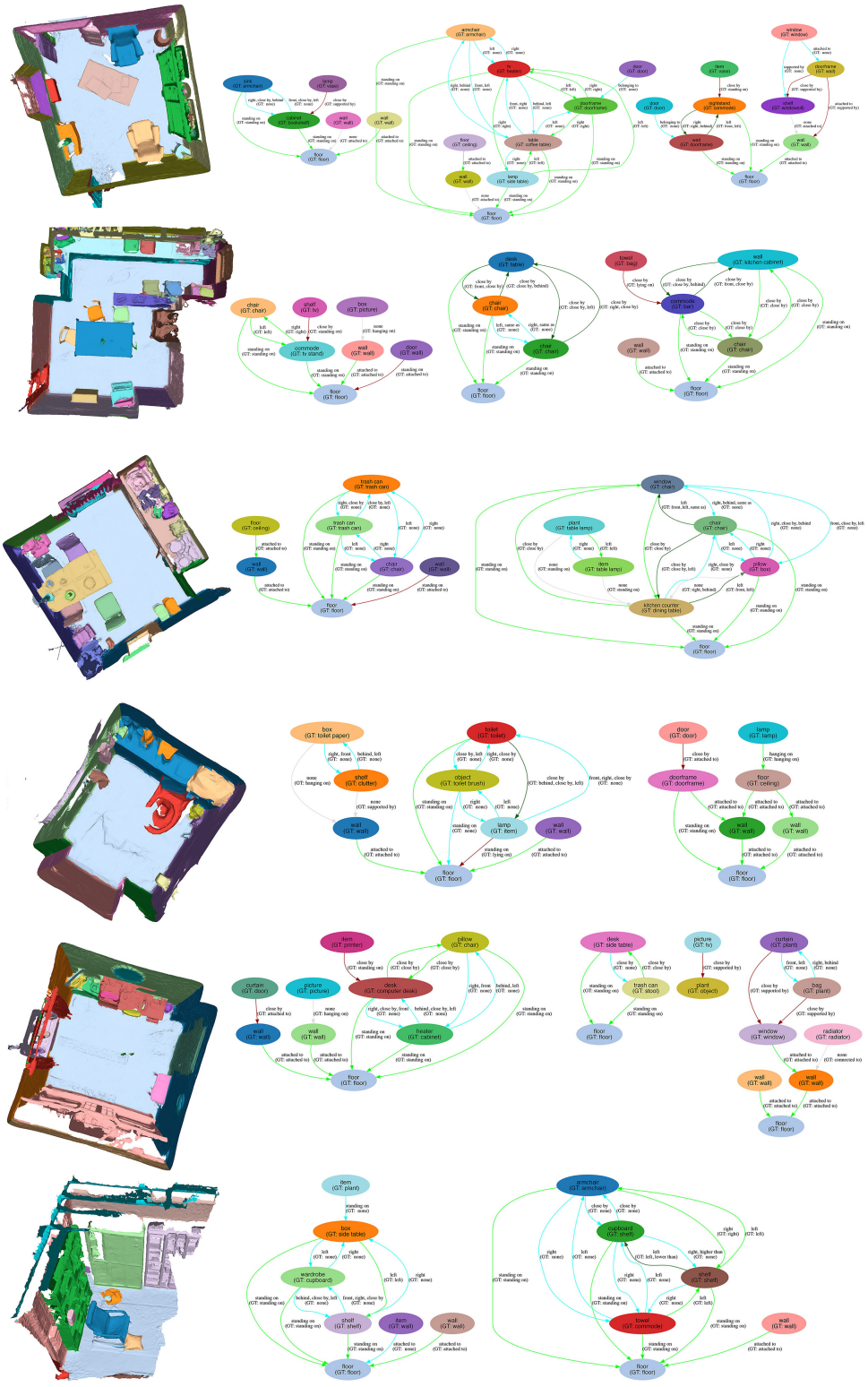


Figure 4.14. **Qualitative results of our scene graph prediction** (best viewed in the digital file). *Light green*: correctly predicted edges, *dark green*: partially correct edges, *blue*: false positives – missing ground truth, *red*: miss-classified edges, *gray*: false negatives – wrongly predicted as *none* when the ground truth is a valid relationship.

4.6 Graph-to-3D: 3D Scene Generation and Manipulation

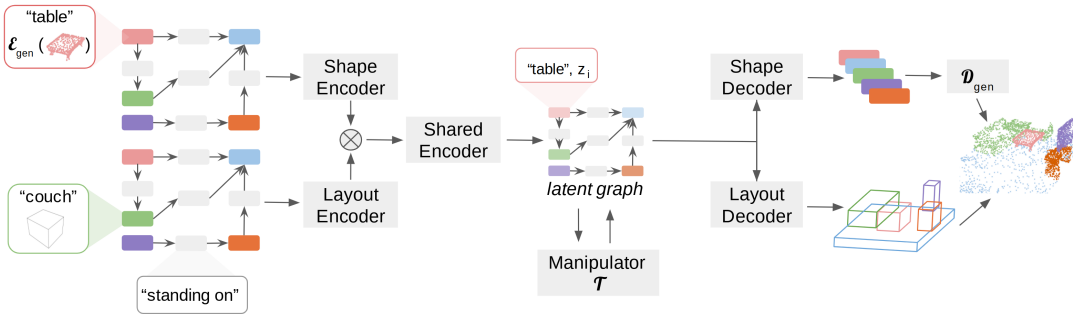


Figure 4.15. **Overall Graph-to-3D architecture**. Our model generates a 3D scene as a set of 3D bounding boxes and object shapes for a given scene graph. To this end, we use a scene graph variational Auto-Encoder with two parallel GCN encoders for shape and boxes. The latent information from the box and shape component is combined through a shared encoder. The final 3D scene is obtained by sampling from the shared latent distribution and combining the predictions from the two GCN decoders for 3D boxes and shapes. We further use a GCN manipulator to support user modifications to the scene. [©2021 IEEE]

Having annotated real 3D world environments with scene graphs, we can next investigate the novel problem of 3D scene generation from an input scene graph, in a fully learned manner. Thus, given a scene graph $\mathcal{G} = (\mathcal{N}, \mathcal{R})$, where nodes $n_i = c_i \in \mathcal{C}$ are semantic object categories and edges $r_{ij} \in \mathcal{R}$ are semantic relationship categories with $i \in \{1, ..., N\}$ and $j \in \{1, ..., N\}$, the goal is to generate a 3D scene S as described by the graph. The 3D scene here is represented $S = (\mathcal{B}, \mathcal{S})$ as a set of per-object 3D bounding boxes $\mathcal{B} = \{b_0, ..., b_N\}$ and shapes $\mathcal{S} = \{s_0, ..., s_N\}$. Our model is based on a variational scene graph Auto-Encoder, inspired by [96] on 3D layout generation for the purpose of image synthesis. However, while [96] use a retrieval-based approach to populate the generated boxes with object shapes, we learn layouts and shapes jointly via a shared latent embedding, as these are two inherently cohesive tasks that should support each other. Additionally, we enable the related task of scene manipulation, using *the same* learned network. Similarly to the task from Section 4.3, given a 3D scene and its respective scene graph, a user can modify the scene by making changes in the graph, such as adding new nodes or changing relationships. In 3D we do not need to learn object removal as this can be simply achieved by discarding the respective shape and box.

Figure 4.15 shows the pipeline of the proposed method. First, we encode the layout and shapes conditioned on scene graphs via a layout $\mathcal{E}_{\text{layout}}$ and shape $\mathcal{E}_{\text{shape}}$ encoder, Section 4.6.2. Further, a shared encoder $\mathcal{E}_{\text{shared}}$ combines features from $\mathcal{E}_{\text{layout}}$ and $\mathcal{E}_{\text{shape}}$, Section 4.6.3. The shared embedding is then processed by a shape decoder $\mathcal{D}_{\text{shape}}$ and a layout decoder $\mathcal{D}_{\text{layout}}$ to obtain the boxes and shapes of the reconstructed 3D scene. Finally, the manipulation network $\mathcal{T}$ enables user-induced changes in the scene. In the following sections, we describe each of these components in detail.

4.6.1 Data preparation

For the purpose of learning object poses and shapes, we require canonical pose annotations which are not present in the 3RScan dataset or our 3DSSG extension with scene graphs. We thus carried out an efficient semi-automatic annotation framework to obtain canonical oriented (tight) bounding boxes for each instance. We model the oriented boxes with 7 degrees-of-freedom (7DoF) – 3 for size, 3 for translation as well as 1 parameter for the rotation around the z-axis – since the majority of objects is supported horizontally by a planar surface. We use volume as criteria to optimize the rotational parameter, motivated by the fact that the oriented bounding box should ideally fully enclose the object while possessing minimal volume. First, for every object we extract the respective point set p. Then, we rotate the points along the z-axis incrementally using angles α in the interval $[0, 90[$ degrees, with a step size of 1 degree, $p_t = R(\alpha)p$. At each step, we extract the axis-aligned bounding box from the rotated point set p_t, by calculating the extrema of all point coordinates along each axis. We then estimate the area of the bounding box projected in bird's eye view, obtained via an orthogonal projection onto the ground plane. Note that this is equivalent to the minimum volume criteria in a 7DoF scenario. We then label the rotation $\hat{\alpha}$ having the smallest box top-down view area (*c.f.* appendix D for more details). Having obtained the optimal box, we can easily extract the final box parameters: such as the width w, the length l, the height h, the rotation $\hat{\alpha}$ together with the centroid (o_x, o_y, o_z).

Note that for each computed oriented bounding box, we are still left with ambiguity – there are four possible options regarding the object's facing direction. Hence, for objects with two or more vertical axes of symmetry, *e.g.* tables, we automatically define as front the largest dimension (among length and width). This rule is in accordance with the canonical pose definitions from ShapeNet [10], therefore it facilitates learning transfer between different datasets. For other objects such as sofa or chair, we annotate the facing direction manually, leading to 4.3k manually annotated instances in total.

Finally, due to impartial scans in real world reconstructions, we observed misalignments in the bounding boxes (originally obtained from the scene point clouds). Objects are oftentimes detached from their supporting surfaces. For instance a chair with highly reflecting legs leads to a "flying" box which is not touching the floor. We approach this problem by using the support relationships from 3DSSG to detect such inconsistencies. We identify "flying" objects that have a distance of more that 10cm from their support, and adjust the respective bounding boxes, such that they touch the upper level of the support parent. In the case of planar support such as floor, we utilize the RANSAC [29] algorithm to fit a plane in a circular neighbourhood region around the object and fix the height h and centroid parameter o_z of the object box such that it touches the calculated plane.

4.6.2 Encoding a 3D scene

Our network consists of two separate graph encoders for layout $\mathcal{E}_{\text{layout}}$ and shapes $\mathcal{E}_{\text{shape}}$. The layout encoder $\mathcal{E}_{\text{layout}}$ is essentially a GCN that takes the *augmented* scene graph $\mathcal{G}_b$ – where

each node $n_i = (c_i, b_i)$ is extended with the bounding box b_i of each object – and results in a per-node output feature vector $f_{b,i}$, where $f_b = \mathcal{E}_{layout}(\mathcal{G}_b)$.

When generating an object shape in a scene, it is important to consider contextual consistency with the other objects. As an example, one would expect a dining chair to co-occur with a dining table, while an office chair is more likely to be found close to a desk. Thus, we employ a GCN to infer globally consistent object shapes, instead of sampling the shapes independently via a standard shape Auto-Encoder. Learning a GCN Auto-Encoder on shapes, *e.g.* point clouds – similarly to the bounding boxes – is considerably more difficult due to the uncontinuous output space. Therefore, we instead propose to learn shape generation leveraging a latent shape space in canonical poses. This latent space can be obtained via shape generative models which consist of an encoder $\mathcal{E}_{gen}(\cdot)$ and a decoder $\mathcal{D}_{gen}(\cdot)$, such as an Auto-Encoder or Auto-Decoder network [39, 106]. Following the same formulation as in the layout counterpart, we create the augmented scene graph $\mathcal{G}_s$ where each node $n_i = (c_i, e_i^s)$, and $e_i^s = \mathcal{E}_{gen}(s_i)$. Interestingly, as the method consumes latent codes it becomes agnostic to the shape representation. In our evaluations, we experiment with AtlasNet [39] and DeepSDF [106] as state-of-the-art models for namely point cloud and SDF generation. Please refer to appendix D for more details on AtlasNet and DeepSDF. The GCN-based shape encoder $\mathcal{E}_{shape}$, is fed with $\mathcal{G}_s$ to obtain per-node latent shape features $f_s = \mathcal{E}_{shape}(\mathcal{G}_s)$.

4.6.3 Shape and layout communication

We introduce a shared encoder $\mathcal{E}_{shared}$ to foster communication between the inherently related tasks of layout and shape generation. Thereby, $\mathcal{E}_{shared}$ takes as input the concatenation of the result features from both encoders and gives a shared feature as $f_{shared} = \mathcal{E}_{shared}(f_{bs}, \mathcal{R})$ with $f_{bs} = \{f_{b,i} \oplus f_{s,i} \mid i \in (1, ..., N)\}$. The shared features f_{shared} are then fed to a network, implemented as an MLP, to obtain a shared posterior distribution (μ, σ) under a Gaussian prior, where μ and σ are namely the mean and variance. We sample z_i from this distribution and feed the result to the respective layout and shape decoders. To obtain z_i at training time, given that sampling is naturally not differentiable, we apply the commonly used re-parameterization trick.

4.6.4 Decoding the 3D scene

We harness two GCN-based decoder networks whose goal is to learn a mapping from the shared latent representation of the scene objects, as well as the semantic scene graph, to the fully-learned reconstructed 3D scene. The layout decoder $\mathcal{D}_{layout}$ is a GCN network followed by two parallel MLP branches, which predict namely $b_{-\alpha,i}$ (box location and size) and α_i (angle). Concretely, $\mathcal{D}_{layout}$ consumes a set of per-node sampled latent vectors z within the learned distribution, together with the semantic graph $\mathcal{G}$. Its output are the associated object bounding boxes $(\hat{b}_{-\alpha}, \hat{\alpha}) = \mathcal{D}_{layout}(z, \mathcal{C}, \mathcal{R})$. Similarly, the shape decoder $\mathcal{D}_{shape}$ is fed with per-node sampled latent vectors z and the graph $\mathcal{G}$, to obtain the final shape encodings

$\hat{e}^s = \mathcal{D}_{\text{shape}}(z, \mathcal{C}, \mathcal{R})$. The architecture follows the structure of $\mathcal{D}_{\text{layout}}$, with the difference that the GCN here is followed by a single MLP.

Having obtained the set of 3D bounding boxes and shapes, one can finally generate the full 3D scene. Thus, every shape encoding is translated into the respective shape representation using the decoder module of the shape generative model at hand $\hat{s}_i = \mathcal{D}_{gen}(\hat{e}^s_i)$. Each shape $\hat{s}_i$ is then transformed from its canonical pose to scene coordinates, utilizing the obtained 3D bounding box parameters $\hat{b}_i$.

4.6.5 Manipulation network

We extend the Graph-to-3D model with a manipulation network $\mathcal{T}$, which enables semantic changes in the scene, while keeping some parts unchanged. The alternative of directly (and independently) changing a subset of scene nodes, would lead to a model that is not aware of the unchanged scene parts, and therefore eventually evoke collisions and other inconsistencies. $\mathcal{T}$ is again based on a GCN architecture and receives the shared latent graph $\mathcal{G}_l = (z, \mathcal{C}, \mathcal{R})$ (with nodes $n_i = (c_i, z_i)$) as obtained from the encoder networks. As a first step, we augment the latent graph with node and edge changes $G_l = (\hat{z}, \widehat{\mathcal{C}}, \widehat{\mathcal{R}})$. Thereby, $\widehat{\mathcal{C}}$ is composed of the original node labels $\mathcal{C}$ as well as the newly added/changed nodes $\mathcal{C}'$. Similarly, $\widehat{\mathcal{R}}$ consists of the original graph edges $\mathcal{R}$ together with the new in-bound and out-bound edges $\mathcal{R}'$ of $\mathcal{N}'$. In addition, among the existing relationships $\mathcal{R}$, some predicate labels are modified according to the user input. Note that we do not have any corresponding latent representations for the changed nodes $\mathcal{N}'$, as they are externally produced. We instead pad z'_i with zeros to compute $\hat{z}_i$. For a given semantic change, there can be many possible outputs, regarding the object shape, size and location. To model the continuous one-to-many nature of the output space, we introduce stochasticity by concatenating the changed nodes $\hat{z}_i$ with samples z^n_i from a normal distribution with zero mean and unit standard deviation. The unchanged nodes are concatenated with a vector of zeros instead. The outcome of $\mathcal{T}$ is a set of transformed per-node latent vectors $z_{\mathcal{T}} = \mathcal{T}(\hat{z} \oplus \hat{z}^n, \widehat{\mathcal{C}}, \widehat{\mathcal{R}})$, as illustrated in Figure 4.16. Afterwards, the latents predicted by $\mathcal{T}$ for the modified nodes are plugged back into the original latent graph G_l, to encourage changes which are consistent with the original unchanged nodes. Finally, the changed latent graph is fed to the respective decoders to synthesize the updated scene. During inference, the node and edge changes are induced manually, based on a user's input. At training time, the user input is simulated by making random augmentations to the original scene graph. Essentially, for a given scene graph we randomly either drop a node and discard all its edges, pick a random relationship and change its label, or simply leave the scene graph unchanged.

4.6.6 Training objectives

To train Graph-to-3D on the unchanged nodes, including the generation mode and unchanged nodes in manipulation, we optimize a reconstruction term

$$\mathcal{L}_r(\mathcal{N},\mathcal{R}) = \frac{1}{N}\sum_{i=1}^{N}(||\hat{b}_{-\alpha,i} - b_{-\alpha,i}||_1 + CE(\hat{\alpha}_i,\alpha_i) + ||\hat{e}_i^s - e_i^s||_1), \tag{4.11}$$

combined with a Kullback-Leibler divergence term

$$\mathcal{L}_{KL} = D_{KL}(\mathcal{E}(z|\mathcal{G},\mathcal{B},e^s)|p(z|\mathcal{G})), \tag{4.12}$$

where $p(\cdot)$ denotes the Gaussian prior distribution, $\mathcal{E}(\cdot)$ represents the complete encoding network and CE indicates multi-class cross-entropy. We discretize the angles in bins to obtain 24 classes.

Self-supervised learning for modifications

Since inferring a scene from a high-level graph representation is a one-to-many mapping, directly supervising the changed nodes with a standard reconstruction loss, *e.g.* L_1, is not a suitable modeling for the task. Many possible object shapes would satisfy a certain object category, and many possible object constellations would satisfy a certain relationship constraint. Therefore, we propose a novel relationship discriminator D_{box}, which can directly learn to interpret relationships and layouts from data and is, thus, capable of steering the models to learn appropriate changes. We feed D_{box} with the data describing an object pair, namely two boxes, two object labels, as well as their relationship label. The role of D_{box} is then to enforce that the newly generated box will be following the semantic relationship label. As in a common adversarial training, we feed the discriminator with either real or fake (generated) samples, *i.e.* boxes after modification. Thus D_{box} learns to distinguish between real and fake compositions, while the generator tries to produce realistic compositions to fool the discriminator. The loss optimizes the following objective inspired by [38]

$$\begin{aligned}\mathcal{L}_{D,b} = \min_G \max_D [&\sum_{(i,j)\in\mathcal{R}'} \mathbb{E}_{c_i,c_j,r_{ij},b_i,b_j}[\log D_{box}(c_i,c_j,r_{ij},b_i,b_j)] \\ &+ \mathbb{E}_{c_i,c_j,r_{ij}}[\log(1 - D_{box}(c_i,c_j,r_{ij},\hat{b}_i,\hat{b}_j))]].\end{aligned} \tag{4.13}$$

Notice that this discriminator loss is applied to all edges that contain a relationship change or a node addition.

Additionally, we adopt an auxiliary discriminator [105] that operates at the object level and learns to discriminate between shapes. Such discriminator can learn to distinguish if the synthesized shape comes from the underlying shape distribution. Similarly to the object discriminator from Section 4.3, in addition to the GAN loss, we leverage an auxiliary classification loss $\mathcal{L}_{aux}$

$$\mathcal{L}_{D,s} = \mathcal{L}_{aux} + \min_G \max_D [\sum_{i=1}^{N} \mathbb{E}_{c_i,e_i^s}[\log D_{shape}(e_i^s)] + \mathbb{E}_{c_i}[\log(1 - D_{shape}(\hat{e}_i^s))]]. \tag{4.14}$$

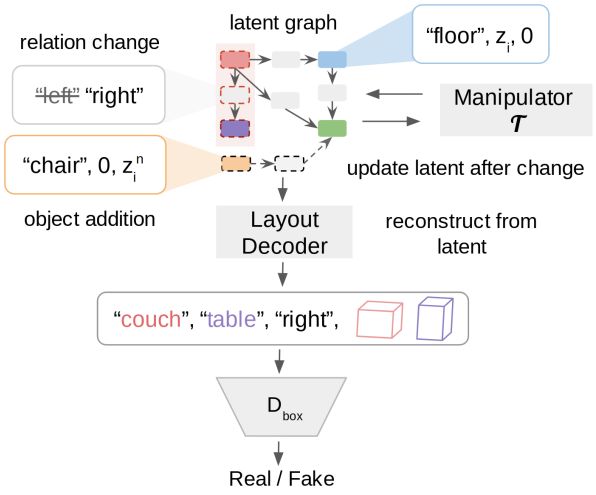


Figure 4.16. **Scene graph modification.** For a provided scene graph we apply changes to the nodes (addition) or edges (relationships). The manipulation network $\mathcal{T}$ takes the latent representations of all nodes and updates the codes for the changed nodes. Edges that underwent changes are then fed to our relationship discriminator which enforces that the box predictions follow the constrains of the node and edge labels. [©2021 IEEE]

to encourage that the synthesized shapes fairly represent their true class. Technically, $\mathcal{L}_{\text{aux}}$ is a cross-entropy loss between the predicted class from the discriminator D_{shape} and the respective true class label c_i.

To summarize, the total loss objective used to train Graph-to-3D is

$$\mathcal{L}_{\text{total}} = \mathcal{L}_r + \lambda_{KL}\mathcal{L}_{KL} + \lambda_{D,b}\mathcal{L}_{D,b} + \lambda_{D,s}\mathcal{L}_{D,s} \tag{4.15}$$

where the λs are the respective loss weights.

4.6.7 Inference

Generation Given a scene graph, we first sample a random vector per-node from the gaussian prior. Then we feed the augmented scene graph (class embeddings and sampled vectors) to the shape and layout decoders to recover a 3D scene.

Manipulation We first encode the input scene given the scene graph (newly added nodes are again sampled from the gaussian prior). We then run $\mathcal{T}$ to update the latent of the changed nodes *w.r.t.* the new graph, decode the scene and add the changes to the input scene.

4.6.8 Implementation details

We use 5 residual layers for each GCN block. In the encoders $\mathcal{E}_{\text{shape}}$ and $\mathcal{E}_{\text{layout}}$, prior to computation, the class categories c_i and r_{ij} are fed to embedding layers, while the shape embedding, bounding box and angles are projected via a linear layer. The shape embeddings

e_i^s have a dimension of 128. Both discriminators are composed of fully-connected layers, where all layers (excluding the last) are followed by batch norm and Leaky-ReLU. For D_{box}, consisting of 3 layers, the last fully-connected layer is followed by a sigmoid. Here the class categories for objects c_i and predicates r_{ij} are fed in one-hot form giving a size of namely 160 and 26. D_{shape} consists of two consecutive layers followed by two branches of fully-connected layers, which end with namely a softmax (for classification) and sigmoid (for discrimination). The architectures of the discriminators are presented in appendix D. We train the model for 100 epochs, using the Adam optimizer with a learning rate of 0.001 and batch size of 8. The training takes one day on a Titan Xp GPU. The loss weights are namely $\lambda_{KL} = 0.1$, $\lambda_{D,b} = 0.1$ and $\lambda_{D,s} = 0.1$.

4.6.9 Evaluation

This section describes the evaluation we used to measure the performance of Graph-to-3D in terms of layout and shape generation/manipulation. This task involves multiple goals and components. We are interested in obtaining realistic shapes and layout configurations, that in addition, fairly represent the labels of the input scene graph.

Evaluation protocol

We evaluate Graph-to-3D on the obtained 3DSSG dataset [141], on the same splits as for the graph prediction task, *i.e.* with 160 classes of objects and 26 classes of relationship predicates. As multiple results are valid for the same input, typical reconstruction metrics, such as L1 norm or Chamfer loss are not ideal, due to their one-to-one comparison between the predictions and the available ground truth. Following [96] we rely on the geometric constraints imposed by the relationship labels to assess if the generated layouts are correct. We evaluate these constraints on each predicted box pair that is labelled with one of the following relations: `left`, `right`, `smaller`, `larger`, `front`, `behind`, `lower`, `higher` and `same`. Note that we exclude other "more semantic" relationships, as they are annotated manually and therefore cannot be captured by a geometric rule, *e.g.* `belonging to`, `leaning against`. More details on the constraint formulation can be found in appendix D.

A quantitative evaluation to assess the quality of the generated shapes, as well as the entire scenes is not straight-forward with the current evaluation metrics. We propose a way to carry out such quantitative evaluation via a cycle-consistency verification. Once we generate the shapes from our models, we feed them to our scene graph prediction network (SGPN) from Section 4.5 to obtain the respective predicted scene graphs. Then, we compare the ground truth scene graphs (*i.e.* input to our scene generation models) against the inferred graphs from SGPN. We motivate this metric by the expectation that plausible shapes and scenes should lead to the same graph prediction as the input graph. Similar evaluation metrics have been proposed for the task of image generation from a semantic map [147], where the input semantics are compared against the predicted semantics from the synthesized image. This comparison is evaluated via the standard top-k recall metric for objects, predicates and relationship triplets, as explained in the SGPN evaluation. Finally, we report a perceptual

user study to assess the global correctness of the scenes, style fitness between the objects as well as validity of the graph constrains.

Baselines

3D-SLN Due to the unavailability of SunCG, we train the closest baseline 3D-SLN [96] to our method on 3DSSG utilizing their official code[3]. Unlike [96] we do not obtain an image output or assume image availability, therefore we discard their render-based refinement. To obtain 3D shapes for 3D-SLN, we employ their retrieval-based approach, in which for every $\hat{b}_i$ we retrieve from 3RScan the most similar object shape from the same class. As in the original 3D-SLN paper, similarity is defined through the norm of aspect ratio distances between the height, width and length dimensions.

Progressive Generation Another model which supports 3D generation and manipulation would be a progressive (auto-regressive) approach. Progressive methods have been explored before for similar tasks, such as PlanIT from [143] for room instantiation based on relational graphs. At each step, a new node is added to the scene. Thereby, a GCN (same as $\mathcal{D}_{layout}$) receives the current scene – as a set of node c and edge r labels, and 3D boxes b – together with a new node n_i to be added. The model then predicts the new box as $b_i = \mathcal{A}(c_i, r_i j, z_i, c, r, z)$. Here z_i represents a randomly sampled noise vector from a normal distribution with zero mean and unit standard deviation. For the novel node n_i, we only feed the object category c_i and its relationships $r_i j$ with existing objects j, while the respective box b_i is masked with zeros. At inference time, in generation mode, the progressive model assumes the first node given, then gradually adds more nodes and their relationships. In manipulation mode, the model receives a current scene and a change to be incorporated, where the boxes of the nodes affected by the change are set to zeros. We train the progressive baseline with varying graph sizes (ranging from 2 to 10), so that it learns to generate the consecutive node for each generation step. We order the nodes hierarchically, according to the graph topology of the support relationships, *e.g.* pillow should be generated after the supporting bed. The disconnected nodes are placed last in order.

Ablations To ablate the effect of utilizing a GCN for the shape generation, we employ a variational autoencoder directly based on AtlasNet. This model is not aware of the scene context, neighbouring objects and connectivity. For a given point cloud s_i we can compute the posterior distribution $(\mu_i, \sigma_i) = \mathcal{E}_{gen}(s_i)$ where (μ, σ) are the mean and log-variance of a diagonal Gaussian distribution. During inference, one can sample from the posterior to generate new shapes. Further, we ablate the effect of sharing between the layout and shape components, by training a variant with separate models for shape (Graph-to-Shape) and layout (Graph-to-Box) generation. Each network follows the same architecture choices for the encoders and decoders, except that $\mathcal{E}_{\text{shared}}$ is not present. Additionally, we train our method without the modification network $\mathcal{T}$, to ablate its influence.

[3] https://github.com/aluo-x/3D_SLN

Method	Shape Representation	left / right	front / behind	smaller / larger	lower / higher	same	total
3D-SLN [96]	–	0.74	0.69	0.77	0.85	**1.00**	0.81
Progressive	–	0.75	0.66	0.74	0.83	0.98	0.79
Graph-to-Box	–	0.82	0.78	0.90	0.95	**1.00**	0.89
Graph-to-3D	AtlasNet [39]	**0.85**	0.79	0.96	0.96	**1.00**	0.91
Graph-to-3D	DeepSDF [106]	0.81	**0.81**	**0.99**	**0.98**	**1.00**	**0.92**

Table 4.5. Scene graph constrains on the **generation** task (higher is better). The total accuracy is computed as mean over the individual edge class accuracy to minimize class imbalance bias. [©2021 IEEE]

Method	Shape Representation	mode	left / right	front / behind	smaller / larger	lower / higher	same	total
3D-SLN [96]	–	change	0.62	0.62	0.66	0.67	0.99	0.71
Progressive	–	change	**0.81**	**0.77**	0.76	**0.84**	**1.00**	**0.84**
Graph-to-Box	–	change	0.65	0.66	0.73	0.74	0.98	0.75
Graph-to-3D w/o $\mathcal{T}$	AtlasNet [39]	change	0.64	0.66	0.71	0.78	0.96	0.75
Graph-to-3D	AtlasNet [39]	change	0.73	0.67	**0.82**	0.79	**1.00**	0.80
Graph-to-3D w/o $\mathcal{T}$	DeepSDF [106]	change	0.71	0.71	0.80	0.79	0.99	0.80
Graph-to-3D	DeepSDF [106]	change	0.73	0.71	**0.82**	0.79	**1.00**	0.81
3D-SLN [96]	–	addition	0.62	0.63	0.78	0.76	0.91	0.74
Progressive	–	addition	**0.91**	**0.88**	0.79	**0.96**	**1.00**	**0.91**
Graph-to-Box	–	addition	0.63	0.61	0.93	0.80	0.86	0.76
Graph-to-3D w/o $\mathcal{T}$	AtlasNet [39]	addition	0.64	0.62	0.85	0.84	**1.00**	0.79
Graph-to-3D	AtlasNet [39]	addition	0.65	0.71	0.96	0.89	**1.00**	0.84
Graph-to-3D w/o $\mathcal{T}$	DeepSDF [106]	addition	0.70	0.73	0.85	0.88	0.97	0.82
Graph-to-3D	DeepSDF [106]	addition	0.69	0.73	**1.00**	0.91	0.97	0.86

Table 4.6. Scene graph constraints on the **manipulation** task (higher is better). The total accuracy is computed as mean over the individual edge class accuracy to minimize class imbalance bias. Top: Relationship change mode. Bottom: Node addition mode.[©2021 IEEE]

Layout evaluation

Table 4.5 reports the geometric constrain accuracy on the generation task, purely based on the synthesized 3D boxes. Graph-to-3D performs better than the baselines on all metrics. Interestingly, it also outperforms the variant with decoupled shape and layout (Graph-to-box) which indicates that the joint learning of shape and layout improves layout generation. Table 4.6 shows the geometric constraint metric on the manipulation task. We report node addition and relationship change separately. One can observe that the progressive model performs best for the node addition task (Table 4.6, bottom), while Graph-to-3D is fairly comparable for relationship changes. This outcome is expected, because the progressive model is explicitly trained for addition, *i.e.* it can process the whole context of the existing scene and add a new node in accordance with this context. Finally, the models that use the

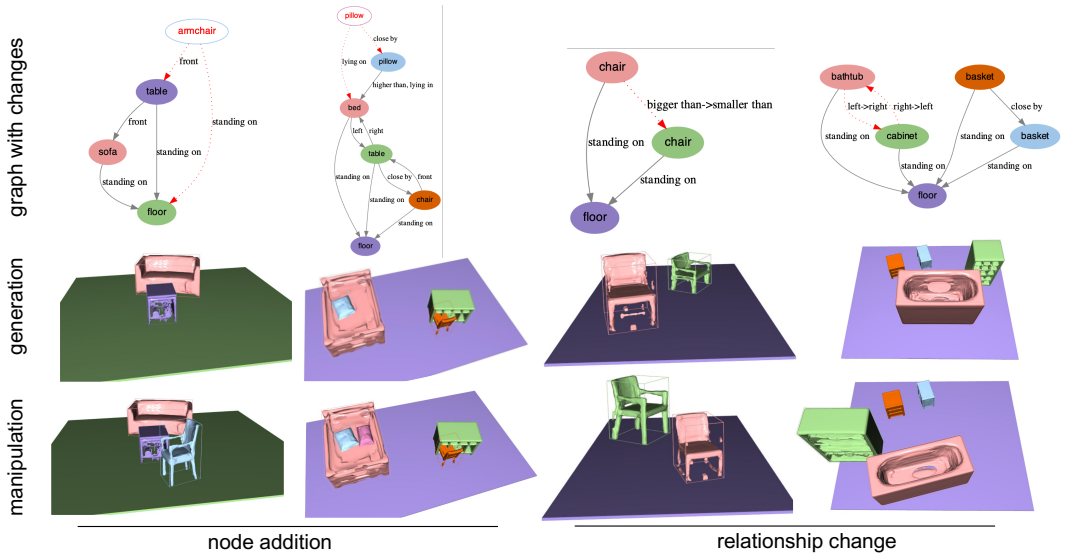


Figure 4.17. **Qualitative results with DeepSDF encoding** of Graph-to-3D on 3D scene generation (middle) and manipulation (bottom), starting from a scene graph (top). Dashed lines reflect new/changed relationship, while empty nodes indicate added objects. [©2021 IEEE]

manipulation network $\mathcal{T}$ perform better than 3D-SLN or the respective Graph-to-3D model without $\mathcal{T}$ on the manipulation task, since $\mathcal{T}$ explicitly models support for changes.

Shape evaluation

Figures 4.17 and 4.18 demonstrate qualitative results from Graph-to-3D trained with DeepSDF encodings and AtlasNet encodings respectively. In both cases we first sample a scene conditioned on a scene graph (top). Then we feed the generated scene to the network, together with a scene graph containing a change, which is then reflected in the 3D scene (bottom). Graph-to-3D understands a diverse set of relationships including support (`lying on`, `attached to`), spatial proximity (`right`, `front`) and attribute comparison (`bigger than`, `same as`). For instance, the model is capable of adding a pillow on the bed (Figure 4.17), or change sofa sizes in accordance with the relationship label (Figure 4.18). The object shapes and sizes fairly represent the class categories in the input scene graph, for both representations.

Next, in Figure 4.19 we demonstrate a few examples that show how Graph-to-3D can leverage scene context on shape generation. We observe that chairs tend to have an office style (middle) when related to a desk, and a more standard style when connected to a dinning table instead (left), or when there is no explicit relationship to a table/desk (right). Moreover, the number of pillows lying on a sofa can affect its style and size, *i.e.* a large sofa vs. a small sofa. These patterns learned from data show a promising advantage of the proposed fully-learned graph-driven method.

Table 4.7 reports the quantitative results as a SGPN prediction consistency on 3D shapes and complete 3D scenes. The object and predicate recall metric are mostly related to shape generation and layout generation respectively. The triplet recall metric considers object and

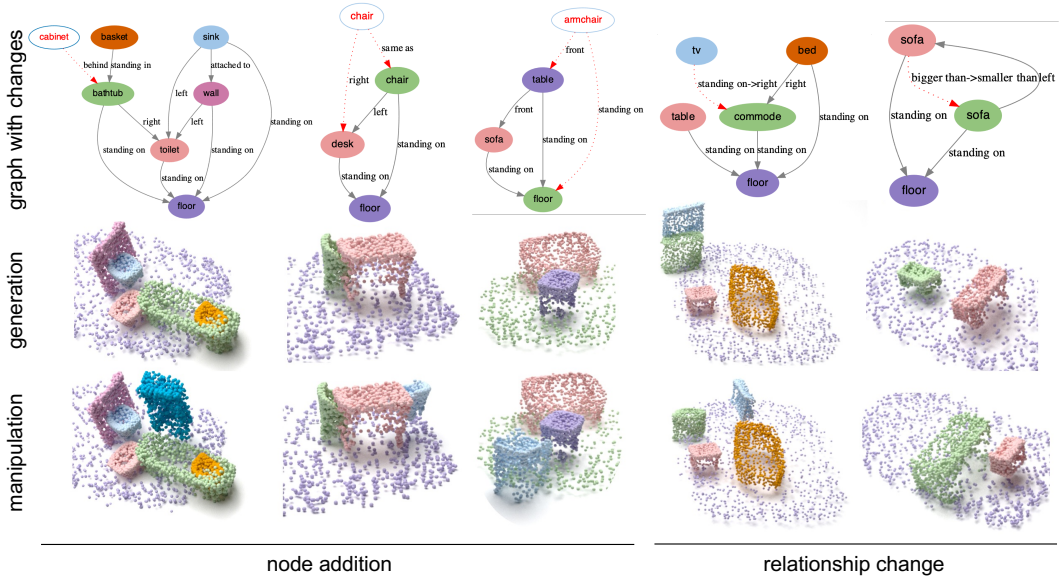


Figure 4.18. **Qualitative results with AtlasNet encoding** of Graph-to-3D on 3D scene generation (middle) and manipulation (bottom), starting from a scene graph (top). Dashed lines reflect new/changed relationship, and unfilled nodes indicate added objects.

predicate labels simultaneously, and is therefore affected by all components. We compare different models, such as retrieval-based 3D-SLN and progressive model, AtlasNet VAE, Graph-to-Box/Shape as well as the shared Graph-to-3D model. The last two models are presented for both AtlasNet and DeepSDF shape encodings. To run the SGPN network on the SDF based results, we first sample a set of points from the generated shapes. For reference we also present the SGPN results on the ground truth scenes (3RScan data). The latter leads to the highest predicate prediction accuracy, which is expected. Interestingly, on shape-related metrics, our Graph-to-3D model gives comparable results to predictions from ground truth data. Models relying on graph processing for shape generation outperform the simple AtlasNet VAE, that is not aware of relationships between the objects. Comparing the shared and disentangled models we observe a consistent performance gain for both the shape and layout-related metrics, meaning the joint layout and shape learning favours both tasks. Generally, the methods based on a point cloud encoding work a bit better than the respective SDF models. This can be explained with the fact that the point sampling from the SDF might not capture the noise in the ground truth point clouds that SGPN is trained with. Finally, to ablate our pose annotations, we also run the Graph-to-Box/Shape variant using shapes in non-canonical poses. Here we still utilize the automatically generated tight bounding boxes, however, the front direction is unknown, leading to four possible rotations. The performance of this model is significantly worse, which confirms the relevance of our direction annotations.

Perceptual study

We carried out a user study with 20 people, each evaluating ≈30 pairs of generated scenes. Each sample in the study contains a scene graph, the 3D-SLN [96] baseline with retrieved

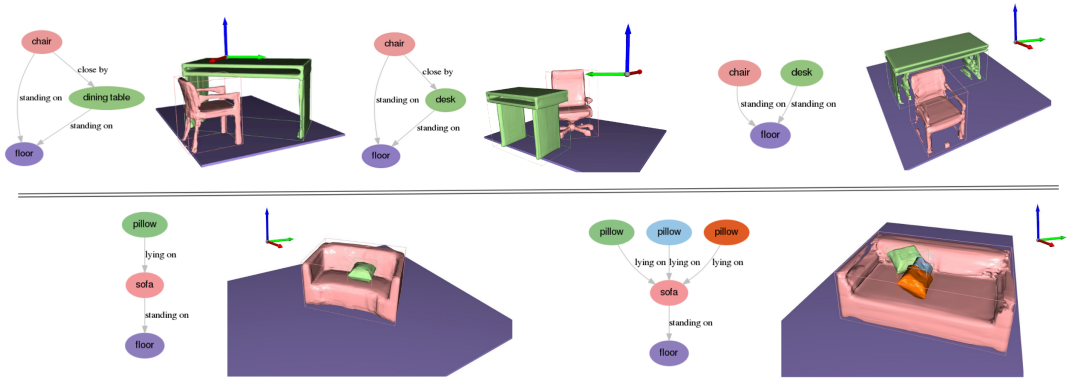


Figure 4.19. **Effect of scene context in scene generation.** *Top:* Connection to a desk makes a chair look like an office chair. *Bottom:* The number of pillows lying on a sofa affects its size and style. [©2021 IEEE]

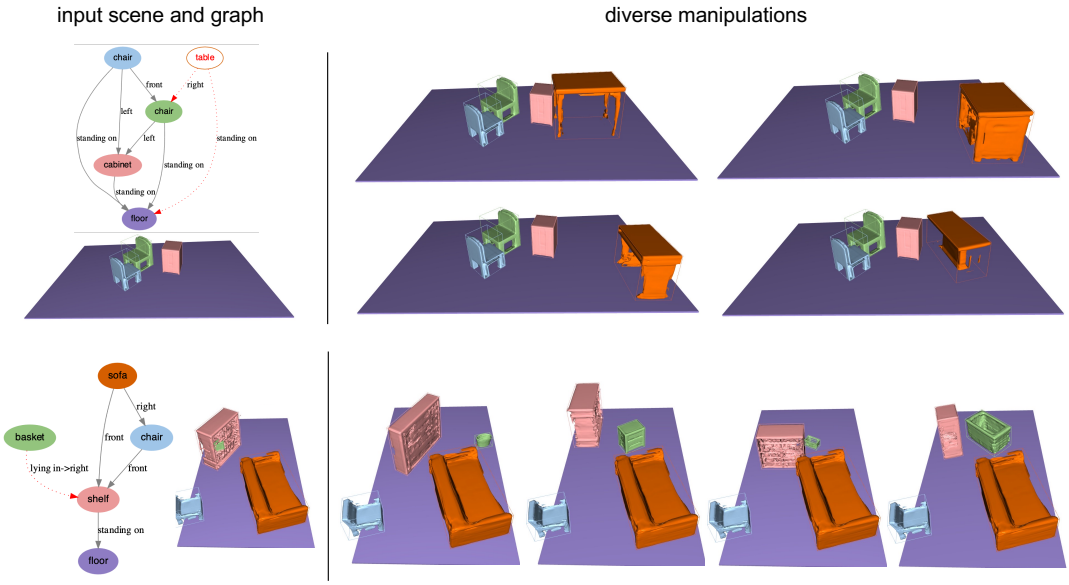
Layout Model	Shape Model	Shape Representation	Recall Objects			Recall Predicate			Recall Triplets		
			Top 1	Top 5	Top 10	Top 1	Top 3	Top 5	Top 1	Top 50	Top 100
3D-SLN [96]	Retrieval	3RScan Data	**0.56**	0.81	0.88	0.50	**0.82**	0.86	0.15	0.57	0.82
Progressive	Retrieval		0.35	0.66	0.79	0.41	0.70	0.82	0.09	0.40	0.70
Graph-to-Box	AtlasNet VAE	AtlasNet [39]	0.41	0.74	0.83	0.57	0.80	0.88	0.08	0.46	0.77
‡Graph-to-Box	‡Graph-to-Shape		0.39	0.68	0.77	0.55	0.79	0.88	0.05	0.35	0.69
Graph-to-Box	Graph-to-Shape		0.51	0.81	0.86	0.57	0.80	0.88	**0.23**	0.63	0.84
Graph-to-3D			0.54	**0.84**	**0.90**	**0.60**	**0.82**	**0.90**	0.21	**0.65**	**0.85**
Graph-to-Box	Graph-to-Shape	DeepSDF [106]	0.47	0.74	0.83	0.57	0.80	0.87	0.14	0.57	0.81
Graph-to-3D			0.51	0.80	0.88	0.58	0.80	0.89	0.19	0.59	0.83
3RScan data			0.53	0.82	0.90	0.75	0.93	0.98	0.18	0.61	0.83

Table 4.7. Scene graph prediction accuracy on 3DSSG, using the SGPN model from [141], measured as top-k recall for object, predicate and triplet prediction (higher is better). ‡Model trained with non-canonical objects, exhibiting significantly worse results. [©2021 IEEE]

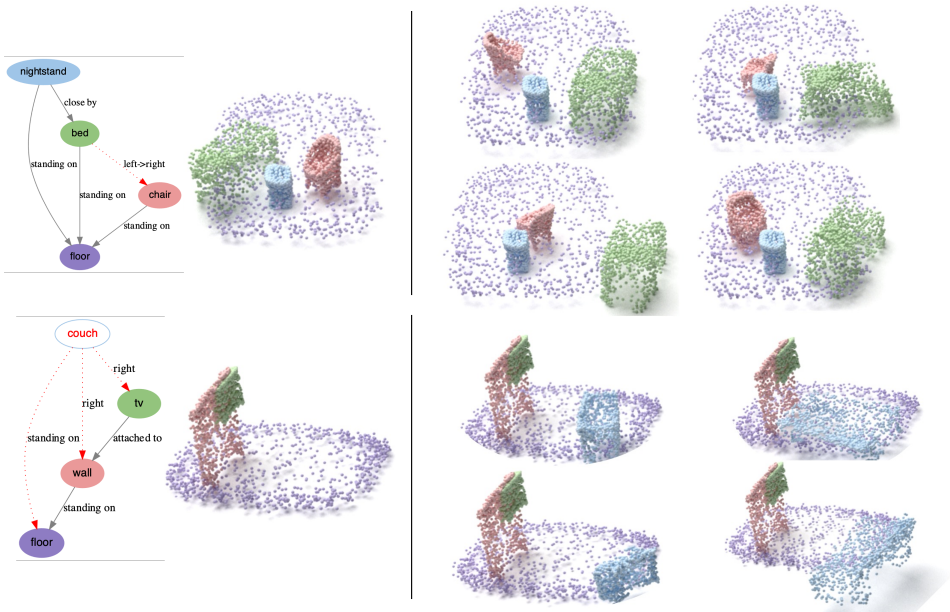
Method	Shape Model	Shape Representation	Generation				Manipulation			
			Size	Location	Angle	Shape	Size	Location	Angle	Shape
3D-SLN [96]	Retrieval	3RScan Data	0.026	0.064	11.833	**0.088**	0.001	0.002	0.290	0.002
Progressive	–		0.009	0.011	1.494	–	0.008	0.008	1.559	–
Graph-to-Box	Graph-to-Shape	AtlasNet [39]	0.009	0.024	1.869	0.000	0.007	0.019	2.920	0.000
Graph-to-3D			**0.097**	**0.497**	**20.532**	0.005	**0.037**	**0.061**	**14.177**	0.007
Graph-to-Box	Graph-to-Shape	DeepSDF [106]	0.009	0.024	1.895	0.011	0.005	0.019	3.391	0.014
Graph-to-3D			0.091	0.485	19.203	0.015	0.015	0.035	9.364	**0.016**

Table 4.8. Comparison on diversity results (std) on the generation (left) and manipulation tasks (right), computed as standard deviation over location and size in meters and angles in degrees. For shape we report the average chamfer distance between consecutive generations.[©2021 IEEE]

shapes from ShapeNet and our Graph-to-3D model with shared layout and shape. To avoid human bias, the scenes are given anonymously and in random order. The users were asked to rate each scene in the range 1-7 (7 is best) on three different aspects 1) global correctness, 2) functional and style fitness between objects and 3) correctness of graph constraints. The results for 3D-SLN are namely 2.8, 3.7, 3.6, while Graph-to-3D reports 4.6, 4.9, 5.4. Our method was preferred in namely 72%, 62%, and 68% of the cases.



(a) DeepSDF encodings



(b) AtlasNet encodings

Figure 4.20. Diverse generation of shapes and layout during manipulation. Given an input graph and correspondingly generated scene (left), we obtain diverse results (right) for the added or changed objects.

Diversity

In Table 4.8 we quantitatively evaluate diversity of the generated and manipulated scenes. For the bounding boxes, we measure the standard deviation among 10 samples resulting from the same input. This metric is computed separately as mean over size in meters, translation in meters and rotation angle in degrees. For the shape diversity, we measure the mean Chamfer distance between these 10 samples. We observe that the progressive model performs worst

in terms of diversity for both generation and manipulation. The VAE-based models instead, result in more interpretable diversity values, which are larger for object position than for size. Both shared models (Graph-to-3D) exhibit higher diversity in layout. Regarding shape diversity, the two Graph-to-3D shared models perform better for manipulation, yet, our results for generation are worse. As a reminder, the baseline's shape retrieval method is based on bounding box similarity. One intuition for the more diverse shapes is that, a slight diversity in the box can lead to significant changes in shape, as two completely different shapes can emerge from two similar boxes.

Additionally, We demonstrate qualitatively the capabilities of Graph-to-3D in generating a diverse set of manipulated scenes under the same graph, in Figure 4.20. We provide results corresponding to both shape generative models, *i.e*. AtlasNet and DeepSDF (*c.f*.Figure 4.20 a) and b)). In this experiment, we first generate a scene given an input scene graph. Subsequently, we apply changes in the scene graph, such as object additions and relationship changes, and let the model run multiple times to incorporate these changes. Notably, Graph-to-3D is capable of incorporating diverse manipulations for the same input, considering 3D shape, location, size and rotation.

4.7 Conclusions and future work

In this chapter we presented our methodology for generating and editing scenes, using scene graphs as interface.

We first introduced the novel task of semantic image manipulation using scene graphs. Thereby, we proposed SIMSG – a model that does not require pairs of original and modified images for training, thanks to our novel training strategy based on a reconstruction proxy task. The resulting system provides a way to change either the objects, their appearance or their relationships by directly interacting with the nodes and edges of the scene graph. SIMSG achieves compelling evidence for its ability to support high-level modification of natural images. Nonetheless, generating images with high resolution from a scene graph is still an open problem. While the image generation quality increases when employing more recent decoders such as SPADE – even for higher resolutions – the SGN's capability for layout generation degrades. This can be partially attributed to the self-supervised inference of object masks, leading to a weak signal for higher resolutions; and the dominance of the image generation task over layout inference.

Further, to pave the way towards similar tasks in 3D, we extracted 3DSSG, a large-scale 3D scene graph dataset with semantic relationship annotations for real-world 3D reconstructions. The dataset has been released to facilitate future research. We leveraged 3DSSG to train a GCN based graph prediction network (SGPN) from 3D scenes that is capable of predicting object semantics as well as the relationships between objects. Thereby, we illustrated that using graph convolutions for scene graph processing leads to better performance than simply considering each object pair.

Finally, we proposed Graph-to-3D, a novel model for fully-learned 3D scene generation from scene graphs, which is simultaneously able to conduct scene manipulation. Thereby, we utilized the predictions from SGPN to evaluate if the generated scenes are in consensus with the input scene graph information. We observed that joint learning of shape and layout led to improvements for all metrics, confirming our motivation for exploring the potential of fully generative models as an alternative to retrieval-based approaches for shape. We show that Graph-to-3D can be trained with different shape representations – without changes in the architecture – here demonstrated with point clouds and implicit functions (SDFs). Our evaluations on visual quality, semantic constrains, perceptual study and diversity have exhibited compelling results on the generation and manipulation task.

Future work could, first, be dedicated to improving certain aspects of the proposed models. For instance, SIMSG could be improved by investigating architectural modifications that enable it to work with higher image resolutions; or by exploring pose-appearance decoupling strategies to enhance the preservation of visual features when objects are deformed. SGPN can be adapted to accommodate larger scene graphs, or process a raw scene end-to-end, without need for class-agnostic segmentation. Graph-to-3D can be adjusted to additionally generate a texture for each instance, combined with scene graph attributes that describe texture-relevant properties. In addition, for the future it would be of high interest to design graph-based models for translating between different domains such as image, text and 3D

scans, using scene graphs as a latent representation as it is naturally compatible with all domains.

5 Publications not Discussed in this Dissertation

This chapter presents research works performed during the thesis time frame that are not thoroughly discussed in this dissertation, which are however relevant to this research topic.

5.1 Unconditional Scene Graph Generation

So far in this dissertation, we have explored the scene graphs capabilities in synthesizing scenes conditionally, either as generation of whole scenes or manipulation of parts. A natural further step would be to consider unconditional generation of scenes. A model that is capable of generating novel scenes understands common patterns of object constellations and can recognise faulty or unusual configurations. Different from other scene generation works we propose a generative model (SceneGraphGen) to learn a distribution of scene graphs, instead of scenes in their final representation such as images or 3D. The motivation of this choice is to learn the underlying semantic structure of real-world scenes more effectively, which is relevant especially for complex scenes, where current unconditional scene generation models struggle. Once a scene graph is sampled, one can translate it to an actual scene by using a graph-to-scene model of choice, such as [65] or [20]. The contributions of this work include:

1. being the first work that learns an unconditional generative model on semantic scene graphs associated with natural scenes.
2. propose an adaptation to an existing label-less graph generative model [163] to accommodate scene graph data.
3. propose two Maximum Mean Discrepancy (MMD) based metrics to evaluate the novel task at the graph and node level.

Here we provide a brief overview of the underlying model and results, and refer the reader to the respective publication for additional details [33].

5.1.1 Auto-regressive generation model

Given a set of n scene graph samples $\mathbb{G}_s = \{G_s^1, G_s^2, .., G_s^n\}$, which are assumed to represent the distribution of scene graphs $p_{data}(G_s)$, the goal is to learn a generative model from $\mathbb{G}_s$, which can later generate novel scene graph samples. In other words, we want to learn a

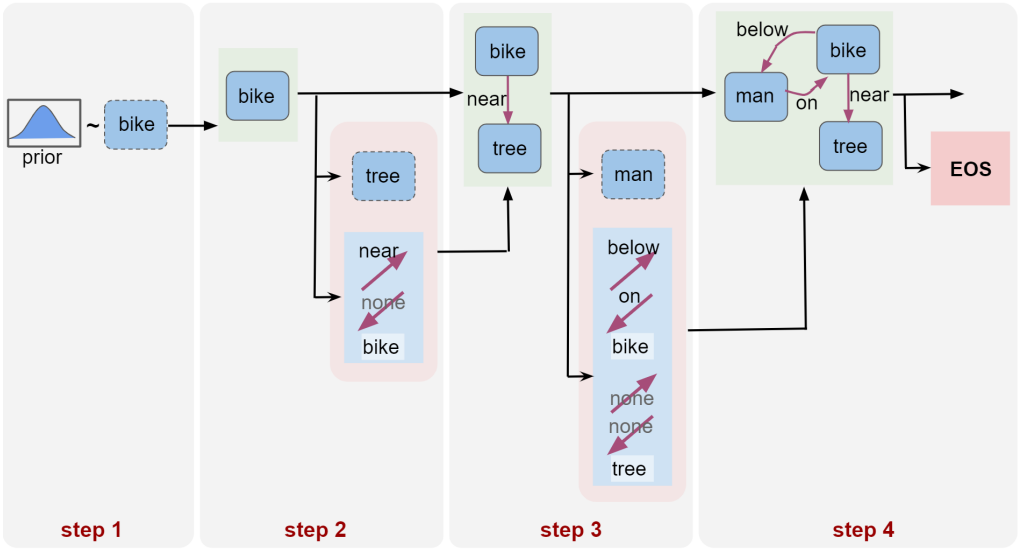


Figure 5.1. Overview of the auto-regressive generation process of SceneGraphGen. In each step, the current graph sequence (green) is taken as input, to generate a new node and a set of connecting edges (red). In the first step, the node is sampled from a prior distribution.

distribution $p_\phi(G_s)$ over scene graphs which is close to $p_{data}(G_s)$. A scene graph sample $G_s = (\mathcal{O}, \mathcal{E})$ follows the same definition as in Chapter 4, consisting of semantic labels for objects and relationship edges. We formulate this task as an auto-regressive model, as it enables a flexible number of nodes and graph connectivity, which are crucial to fairly represent the highly varying scene graphs associated with natural scenes. Inspired by [163] to enable such auto-regressive formulation we represent each scene graph as a sequence $X = (O, E^{to}, E^{from})$, where O denotes a sequence of all objects from the set $\mathcal{O}$, and E represents the sequence of all outbound (to) and inbound (from) edges.[1] Note that the sequence X is itself composed of multiple sequences. For a given node in the sequence we have $X_i = (O_i, E_i^{to}, E_i^{from})$, where O_i is the object node, E_i^{to} and E_i^{from} are the sequence of edges between O_i and each previous node. Thus we translate the task of learning $p_\phi(G_s)$ to learning a sequence distribution $p_\phi(X)$. The probability over sequence X is decomposed into successive conditionals

$$p_\phi(X) = p(X_1) \prod_{i=2}^{n} p_\phi(X_i | X_{<i}). \tag{5.1}$$

$X_{<i} = (X_1, .., X_{i-1})$ depicts the partial scene graph sequence up to step i. We further split each conditional $p_\phi(X_i|X_{<i})$ into three parts for each of the components as

$$p_\phi(X) = p(O_1) \prod_{i=2}^{n} p_{\phi_1}(O_i|X_{<i}) p_{\phi_2}(E_i^{to}|O_i, X_{<i}) p_{\phi_3}(E_i^{from}|O_i, E_i^{to}, X_{<i}). \tag{5.2}$$

SceneGraphGen thus models the complete probability distribution over scene graph sequences $p_\phi(X)$ as described by Eq. 5.2. Thereby, each component is aware of the sequence

[1] Note that a set and a sequence here are not equivalent, even though they essentially contain the same items. A sequence is defined by the notion of ordering as multiple permutations of a set are possible to obtain a sequence. We refer the reader to [33, 163] for a more detailed explanation and formulation.

history, and edge generation is conditioned on the node categories. Moreover, the inbound edges are aware of the respective outbound edges, to avoid a semantic paradox, *e.g.* `A - behind - B` should not co-occur with `B - behind - A`. Here $p(O_1)$ is an assumed prior distribution over the first node, which can be obtained from the categorical distribution over the object occurrences. Thereby, to process sequence information, we rely on Gated Recurrent Units (GRU) [12].

The overall generation procedure is depicted in Figure 5.1. We first sample the first object node from the prior distribution of nodes, empirically computed from the training set. In other steps i we use the previous sequence X_{i-1} as input to compute the hidden states using three GRU's corresponding to namely the node and two edge directions. These hidden states are then used to obtain the next node i, as well as a sequence of edges connecting to each previous node j. The new node $\hat{O}_i$ is generated via an MLP by sampling from the object category prediction scores θ_i^O. Similarly, we generate the sequences of edges $\hat{E}_i^{to}$ and $\hat{E}_i^{from}$ using two edge GRUs, from the respective prediction scores $\theta_i^{E^{to}}$ and $\theta_i^{E^{from}}$. The node and sequence of edges are combined to form the next sequence X_i. This process is continued until the node generator outputs an end-of-sequence (EOS) token.

The network is optimized via cross-entropy (CE) between the predicted scores and the ground truth sequence at each step. For a sample scene graph sequence X, the loss is given by

$$\mathcal{L}(X;\phi) = \sum_{i=2}^{n} CE(\theta_i^O, O_i) + \sum_{i=2}^{n}\sum_{j=1}^{i-1} CE(\theta_{i,j}^{E_{to}}, E_{i,j}^{to}) + \sum_{i=2}^{n}\sum_{j=1}^{i-1} CE(\theta_{i,j}^{E_{from}}, E_{i,j}^{from}) \tag{5.3}$$

5.1.2 Result highlights and discussion

To evaluate SceneGraphGen we carried out a set of experiments which measure the performance in scene graph generation, as well as three different applications: unconditional image generation, anomaly detection and scene graph completion. For the scope of this dissertation, we provide a highlight of the scene generation results, which are more relevant for this thesis research topic. We refer the reader to the published paper [33] for a formulation of the proposed MMD metrics, as well as the complete evaluation.

We evaluated our model on the Visual Genome dataset. In absence of a direct baseline, we compared against GraphRNN [163], adapted to generate categorical labels instead of the original binary output, as well as to support directed edges. The quantitative evaluation shows a relative improvement of 5% in the MMD metric and 80% in the MMD graph metric. We also evaluate at the image level, using sg2im [65] to convert the respective generated graphs to images. The FID value reports a relative improvement over GraphRNN by 4.6%. This shows the relevance of conditioning the edge prediction on the categories of the respective object nodes, which is the main difference between the methods.

Figure 5.2 shows a few samples of generated images by converting our generated graphs via the sg2im network. Interestingly, comparing the FID and Inception score results of images generated from our sampled graphs with those generated from ground truth graphs, also using sg2im, we observe very comparable values (a slight dominance in FID of our method by

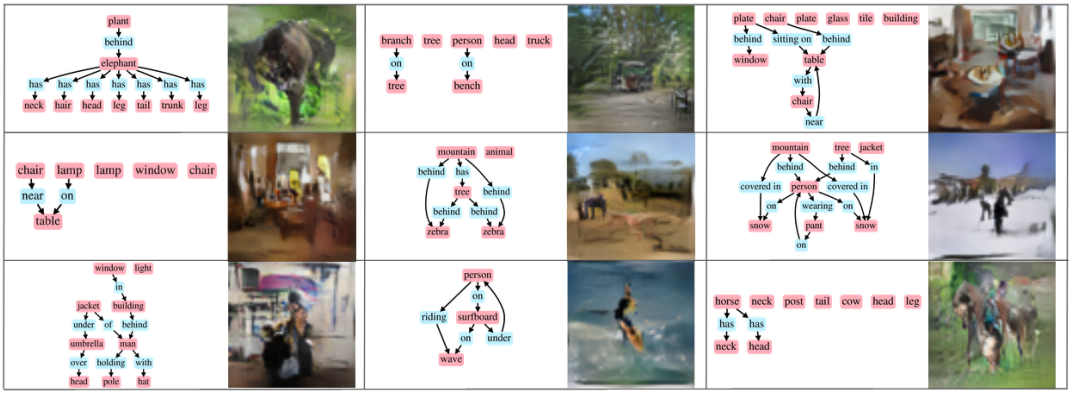
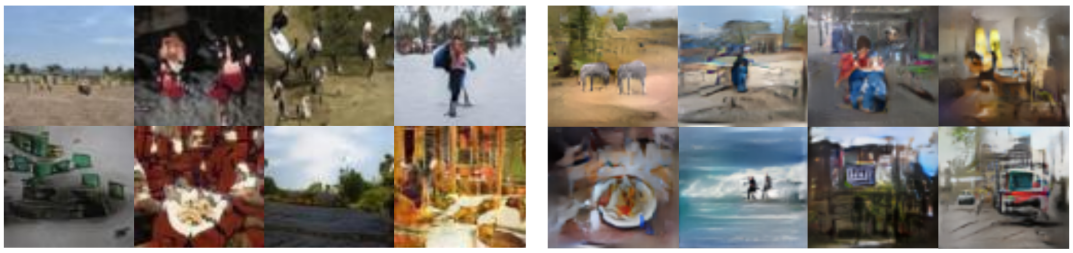


Figure 5.2. Some examples of 64x64 images synthesized using sg2im on the corresponding scene graphs generated by SceneGraphGen. [©2021 IEEE]



a) Unconditional StyleGAN2 **b)** SceneGraphGen graphs + sg2im

Figure 5.3. Some examples of 64x64 resolution images synthesized via **a)** Unconditional StyleGAN [67] trained on Visual Genome (left) **b)** sg2im [65] on scene graphs generated by SceneGraphGen trained on Visual Genome (right). [©2021 IEEE]

2.4%, while ground truth is 5.2% better in Inception score). This shows that SceneGraphGen has learned to generate meaningful graphs, leading to similar image quality as ground truth scene graphs.

We additionally compare our generated images against StyleGAN2, a state-of-the-art model in unconditional image generation [67], trained on the same dataset. Figure 5.3 illustrates results for both methods. We observe that, while StyleGAN2 leads to better FID results, the images resulting from our generated graphs have more well grounded compositions, especially as scenes become more complex. Additionally, comparing the object occurrences against the ground truth test set of Visual Genome, shows that our model is more in line with the ground truth compared to the StyleGAN2 model, with the ground-truth average error of 1.2, compared to 1.4.

We conclude that leveraging scene graphs as an intermediate step for unconditional image generation is a promising future direction. The images synthesized from generated graphs are comparable in quality with those generated from ground truth graphs, and are better grounded semantically compared to conventional unconditional image generation results. We see a natural improvement for the graph-based unconditional image generation as models for generating images from scene graphs get better.

Part III

Conclusion and Outlook

Summary and Findings

6

In this dissertation, we explored compositional representations for scenes, with the aid of scene understanding, to tackle tasks relying on partial or complete scene generation. In the image domain, we essentially employ recent advances in semantic instance segmentation [42] and monocular depth prediction [76] to construct such representations, and thus enhance the given input to enable a modular control on the parts therein. We have shown that having knowledge on the scene components can lead to improved performance, while providing additional functionalities such as relationship changes.

Our object-driven approach for monocular LDI generation (OMLD) has demonstrated superior results – qualitatively and quantitatively – compared to the baselines at the time of publication [21]. As expected, the effect is more drastic in the occluded parts of intermediate (partially occluded) objects, as OMLD takes special care about occluded regions of each individual instance. Further, we have illustrated how the proposed object-aware composition is capable of object removal, which is currently not possible with any other LDI generation work.

We have also shown that decomposing an image via a scene graph, into semantic nodes and edges can be useful for high-level image manipulation. In particular, the graph augmented with neural features and bounding box coordinates per instance provides a flexible tool, in which the user can choose what aspects of an object (location, appearance) to alter or preserve. Making changes in a graph involves less user effort than manipulating pixels in a semantic map [107, 146]. Furthermore, scene graphs enable the additional capability of directly modifying relationships.

Comparing the depth-based and graph-based representations for images, we conclude that, while dense depth information offers the possibility for efficient novel view synthesis, scene graphs come with the advantage for more types of content manipulation. In particular, although it is technically possible to change the order of objects in the scene of a layered depth image – like it is via changes in the edges of the graph – one needs to make sure that there are no collisions occurring in the novel constellations, which essentially can only be enforced through hand-crafted rules. Nevertheless, considering the object removal task – the layered depth representation leads to generally more plausible shapes in the inpainted regions, as it explicitly models amodal appearance of objects.

We explored a similar scene graph representation associated with 3D data, with a focus on obtaining semantically rich relationship labels to describe real indoor scenes. We released the acquired 3DSSG dataset for future research. Thereby, we discussed the advantages of annotating scene graphs in 3D, as the respective 2D scene graphs can be obtained by

simply rendering the 3D counterpart from a certain camera view. We also proposed the first learned method that estimates a scene graph from a point cloud, using 3DSSG for training. In the respective publication [141] we demonstrate that 3D and 2D graphs can be applied for cross-domain retrieval, which is here only discussed in appendix A.

Further, we investigated our final goal of 3D scene generation and editing via a scene graph. We have shown that an augmented 3D graph representation that contains information on 3D bounding boxes, shape encodings and semantic labels is suitable for simultaneous end-to-end synthesis and manipulation of scenes.

Compared to 2D compositional representations, 3D modeling reduces the ambiguity of certain relationships, such as proximity. Exemplary, a certain relative placing of bounding boxes in the image domain can potentially represent both a `front of` and `behind of` relationship, dependent on the object sizes. As expected, we have observed more failure cases in such relationship categories in 2D, compared to 3D. Additionally, in 3D object removal becomes trivial – it is very straight forward to separate the content of a certain object from the rest of the scene, by simply discarding the corresponding points, while in the image space one has to adequately inpaint the regions occluded by this object. Theoretically, it is possible to obtain an image from the generated 3D scene, given a viewpoint. Thereby, a 3D scene would allow for larger viewpoint changes compared to a layered depth structure. On the other hand, synthesizing an image directly on the image domain would in practice lead to a considerably higher quality compared to rendering from synthesized 3D models, thanks to the great recent advances in GAN networks for image generation [67].

7 Future Work

Alongside the potential improvements on the proposed representations and AI models, which are discussed in sections 3.6 and 4.7, here we want to debate possible directions for future research harnessing such scene representations.

This dissertation presents depth-based and graph-based representations separately, due to the large domain gap between the respective datasets, *e.g.* indoor depth datasets vs in-the-wild scene graph datasets. However, it would be an interesting future direction to combine all these components for a more holistic scene understanding and representation, which exploits each individual advantages. For instance, depth information can be useful to disambiguate spatial proximity relationships. One step towards this goal is our generated 3DSSG dataset, that contains all these modalities for training and evaluation of the employed AI models (see Appendix C, rendered scene graphs).

Another interesting direction would be to employ scene graphs as a domain-agnostic modality to translate between 2D and 3D scenes, *i.e.* carry out rendering and inverse rendering via a semantic graph. For instance, one could utilize a unified representation for visual features in 2D and 3D to augment the graph nodes. Similarly, the same system can additionally accommodate natural language, so to obtain 2D/3D scenes from a text description or add a caption to the given scene.

One potential future direction would be to combine a compositional representation at the scene level with one at the object-level. For instance, StructureNet [101] – which employs a graph structure to represent object parts in 3D – would be a fruitful extension to our 3D scene graph, to obtain a graph of graphs. In the light of scene synthesis and editing, such hierarchical model can allow additional customization for the user, by specifying how certain object parts should be, or interpolating between two given object parts.

Despite the high recent interest in image generation from a scene graph, there is little attention in explaining what these networks learn. I believe this would be particularly relevant for such a task as it relies on many components (nodes, edges) and dissecting the model to understand the influence of each part could lead to interesting answers. This analysis could support more effective design choices in the future.

As our acquired 3D scene graph dataset comes with dynamic scenes, it is possible to leverage the scene graphs of multiple scans of the same room to recognize changes. Often this changes are a result of human activity and interaction with the objects in the room, which I believe is an interesting future study direction. This is often related to changes in state attribute, *e.g.* an oven going from `open` to `closed` indicates that someone is potentially cooking.

One possible application in robotics, utilizing the scene graph representations, would be to recognize unusual states that require the robot's action. Exemplary, to identify if a room is messy, one idea is to train an unconditional generative model on scene graphs – similar to the one discussed in Chapter 5 – on a set of graphs corresponding to tidy rooms, and identify outlier graphs at test time as messy.

To conclude, in my opinion there is quite a lot of potential in using the compositions described in this dissertation, or their extended variants, for future research.

Part IV

Appendix

Scene Graphs for Domain-Agnostic Scene Retrieval

A

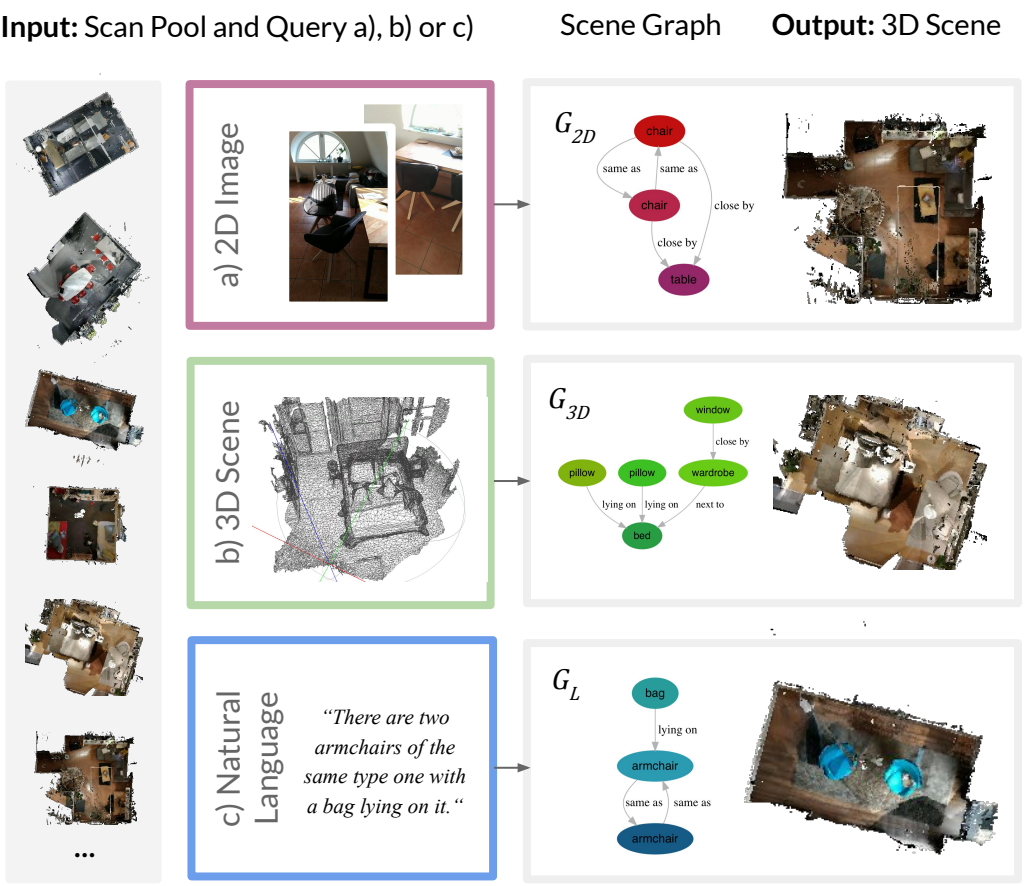


Figure A.1. **Cross Domain 2D-3D Scene Retrieval:** We use scene graphs in our domain-agnostic scene retrieval task to close the domain gap between different modalities, like 2D images and 3D scenes. [©2020 IEEE]

This section illustrates an application of the obtained 3D scene graphs. We exploit the semantic nature of scene graphs in the task of scene retrieval, as a means of communication between different domains. Given a database of 3D scans of scenes, and an image taken at a different time in one of these environments, the goal is to identify the closest match from the database to localize, *e.g.* the robotic agent that is taking the picture. In particular, we consider the scenario of dynamic indoor environments, with potential changes in illumination and object placement, which fairly represents a real life situation in the aforementioned task. We argue that scene graphs are very suitable for this cross-domain task in presence of scene

changes, as they encode semantic information which is less likely to change, whilst serving as a shared domain which can describe both 2D and 3D. Though we only explore with images and 3D scenes, a transfer between other domains such as natural language is technically possible.

In essence, we formulate this task as a database search, based on a set of object classes and relationship triplets. Further, we need similarity metrics to find the most similar scene in the database. Note that though comparing the graphs directly via their graph edit distance is theoretically possible, this leads to a NP-complete problem, which motivates our simplification to multisets of nodes and edges.

To get the similarity of two graphs, a similarity score τ is applied on the corresponding multisets $s(\mathcal{G})$ respectively. In our experiments we explore two different similarity functions: Jaccard $\tau_J(A, B)$, Eq. A.1 and Szymkiewicz-Simpson $\tau_S(A, B)$, Eq. A.2.

$$\tau_J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{A.1}$$

$$\tau_S(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)} \tag{A.2}$$

While the Jaccard coefficient is a widely used metric, the Szymkiewicz-Simpson coefficient is more suitable when the two sets A and B differ considerably in size, which is often the case in a 2D-3D retrieval, with the image graph often being considerably smaller. Our graph matching procedure combines the similarity metric of the object semantics, generic label-less edges $\mathcal{E}$ as well as semantic relationships $\mathcal{R}$ to obtain

$$f(\hat{\mathcal{G}}, \hat{\mathcal{G}}') = \frac{1}{|\hat{G}|} \sum_{i=1}^{|\hat{\mathcal{G}}|} \tau(s(\hat{\mathcal{G}}^{(i)}), s(\hat{\mathcal{G}}'^{(i)}))^1 \tag{A.3}$$

where τ is either the Jaccard or Szymkiewicz-Simpson coefficient and $\hat{\mathcal{G}}$ is defined as the augmented graph $\hat{\mathcal{G}} = (\mathcal{N}, \mathcal{E}, \mathcal{R})$ where $\mathcal{E}$ are binary edges.

Table A.1 and A.2 report two different scene retrieval tasks.[2] The goal is to match either an image (Table A.2) or a 3D scan (Table A.1) with the most similar indoor scene from a database of 3D reference scans from the validation set of 3RScan. For this purpose, we predicted scenes graphs for all the 3D scans, as well as for the images. The image (2D) graphs are obtained by rendering the predicted 3D graphs as described in Section 4.4. Note that the query image or 3D scan is always recorded at a different time (rescan in 3RScan) from the scans in the databaze (reference scans in 3RScan). We compute the scene graph similarity between each query and the pool of reference scans. We then order the matches by their similarity and report the top-n metric, *i.e.* the rate of the true positive assignments, placed among the top-n matches from our algorithm. In our experiment, we either use ground truth or predictions

[1] we define f_S and f_J to use τ_S and τ_J respectively.
[2] In the tables $\hat{\mathcal{G}}$ is replaced with $\mathcal{G}$ to simplify notation

for the query and target graphs (indicated in the Graph-column in Table A.1 and A.2). To decouple the effect of the different similarity functions from the graph prediction accuracy, we first evaluate $\tau_J(A, B)$ and $\tau_S(A, B)$ using ground truth graphs. As expected, using the Szymkiewicz-Simpson coefficient leads to better results in 2D-3D matching, whereas for 3D-3D matching the performance of both coefficients is on par. Moreover, adding semantic edges to the graph matching – in addition to simple binary edges – improves the accuracy. The tables also confirm that our predicted graphs ② achieve better performance than the baseline model ①.

	Graph	Top-1 ↑	Top-3 ↑	Top-5 ↑
$\tau_S(s(\mathcal{N}_{3D}), s(\mathcal{N}_{3D}))$	GT	0.86	0.99	1.00
$f_S(\mathcal{G}_{3D}, \mathcal{G}_{3D})$	GT	0.96	1.00	1.00
$\tau_J(s(\mathcal{N}_{3D}), s(\mathcal{N}_{3D}))$	GT	0.89	0.95	0.95
$f_J(\mathcal{G}_{3D}, \mathcal{G}_{3D})$	GT	0.95	0.96	0.98
$\tau_J(s(\mathcal{N}_{3D}), s(\mathcal{N}_{3D}))$	①	0.15	0.40	0.45
$f_J(\mathcal{G}_{3D}, \mathcal{G}_{3D})$	①	0.29	0.50	0.59
$\tau_J(s(\mathcal{N}_{3D}), s(\mathcal{N}_{3D}))$	②	0.32	0.46	0.50
$f_J(\mathcal{G}_{3D}, \mathcal{G}_{3D})$	②	0.34	0.51	0.56

Table A.1. Evaluation: 3D-3D scene retrieval of changing 3D rescans to reference 3D scans in 3RScan. [©2020 IEEE]

	Graph	Top-1 ↑	Top-3 ↑	Top-5 ↑
$\tau_J(s(\mathcal{N}_{2D}), s(\mathcal{N}_{3D}))$	GT	0.49	0.75	0.84
$\tau_S(s(\mathcal{N}_{2D}), s(\mathcal{N}_{3D}))$	GT	0.98	0.99	1.00
$f_J(\mathcal{G}_{2D}, \mathcal{G}_{3D})$	GT	0.55	0.85	0.86
$f_S(\mathcal{G}_{2D}, \mathcal{G}_{3D})$	GT	1.00	1.00	1.00
$\tau_S(s(\mathcal{N}_{2D}), s(\mathcal{N}_{3D}))$	①	0.17	0.36	0.42
$f_S(\mathcal{G}_{2D}, \mathcal{G}_{3D})$	①	0.10	0.25	0.32
$\tau_S(s(\mathcal{N}_{2D}), s(\mathcal{N}_{3D}))$	②	0.17	0.36	0.41
$f_S(\mathcal{G}_{2D}, \mathcal{G}_{3D})$	②	0.13	0.38	0.42

Table A.2. Evaluation: 2D-3D scene retrieval of changing rescans to reference 3D scans in 3RScan. [©2020 IEEE]

OMLD: Intermediate Results

B

B.1 Layout ablation

Here, we ablate the components of the layout branch (Network B), via a direct comparison of the layout predictions against the ground truth layouts. Table B.1 demonstrates the effectiveness of the added loss components. In particular, the model variant that does not receive a depth prior, leads to considerably less accurate depth. This is an example of performance gain, due to decoupling of a hard task (*i.e.* predicting invisible depth from a single color image) into two simpler tasks (*i.e.* standard depth prediction and RGB-D inpainting). Further, the employed perceptual loss and adversarial loss lead to an improvement for both texture and depth synthesis.

Method	color		depth	
	MPE	RMSE	MPE	RMSE
Base, without input depth pred	21.42	42.94	0.662	1.091
Base with input depth pred	22.64	42.45	0.505	0.993
+ adversarial loss	20.93	41.47	0.495	0.953
+ perceptual loss	**19.40**	**39.89**	**0.482**	**0.919**

Table B.1. **Ablation of the layout prediction (Network B) on the SunCG dataset.** Base refers to the model as introduced in the paper, where only the reconstruction loss is present $\mathcal{L}_r$. The errors are measured for color range $0 - 255$ and depth in meters.

B.2 Layout and object generation

Here we show the intermediate results of OMLD, object completion and layout prediction, *i.e.* the semantic layers prior to the aggregation in LDI layers. Figure B.1 and B.2 provide examples for each dataset. From top to bottom, we provide the input image, the mask scores predicted by Mask R-CNN, (where opacity indicates confidence), followed by the predictions of our network and ground truth. The two bottom rows report the predicted layouts and the respective ground truth layouts. The generated layers represent fairly plausible object shapes and textures, neglecting the color of the occluding objects. For Stanford 2D-3D-S, the collected ground truth contains holes – in particular for layout – however the network learns from the available pixels to regress continuous maps.

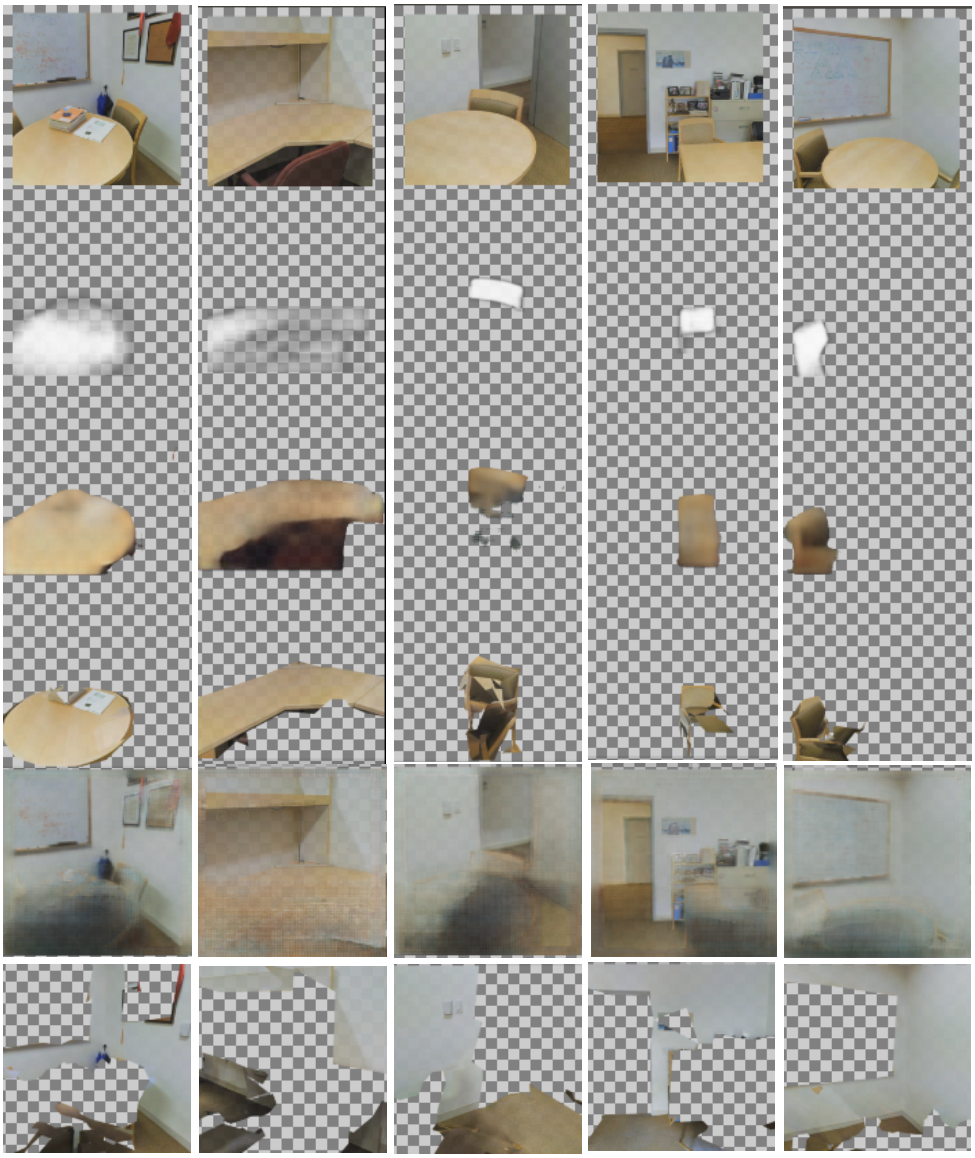


Figure B.1. **RGBA object and layout layers on Stanford 2D-3D.** Input image, instance examples (top to bottom: mask, prediction, ground truth) followed by the layout result.

Figure B.2. **RGBA object and layout layers on SunCG.** Input image, instance examples (top to bottom: mask, prediction, ground truth) followed by the layout result.

C 3DSSG Details and Statistics

C.1 Object state changes

Beside static attributes which do not change over time, such as the color or material of an object, an interesting property of the 3DSSG dataset are dynamic attributes (*e.g.* `messy / tidy`, `open / closed`, `on / off`). Interestingly, one can draw connections between such state properties and human activity. Thus, state changes could potentially be useful in providing information about activities that might have happened in a particular environment. Figure C.1 shows a few scene tuples captured at different time steps in which the state of certain objects has changed.

Figure C.1. Example scenes at two different times where object states have changed. *Left:* `toilet seat` is `down` and then `up` - someone might have used the toilet, *Center:* `floor` went from `messy` to `tidy` - someone might has cleaned the room, *Right:* `bed` went from `tidy` to `messy` - someone might have slept in the bed.

C.2 Rendered 2D graphs

Figure C.2 illustrates 2D scene graphs of the 3DSSG dataset, associated with rendered views of the 3D scene. The rendered graphs naturally come with depth and dense semantic instance masks, which are not present in current 2D scene graph datasets currently available. Visual Genome [73] for instance, one of the most common scene graph datasets in the image domain, only provide bounding box annotations. We believe dense annotations would be relevant to facilitate certain tasks such as image generation.

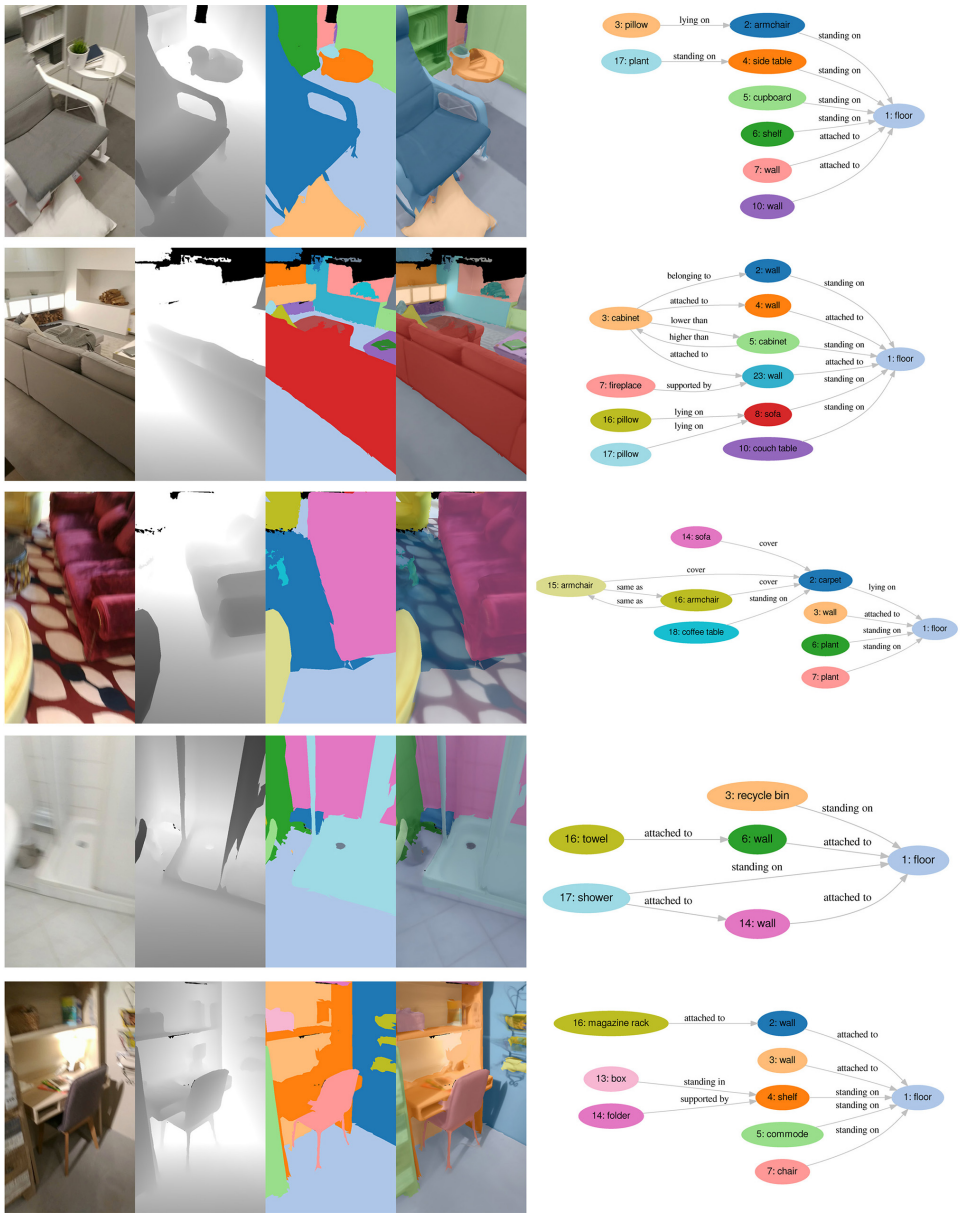
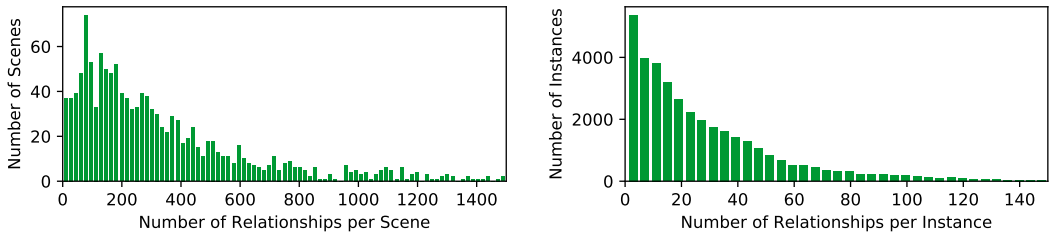


Figure C.2. Rendered 2D graphs with a small subset of the relationships from our newly created 3D semantic scene graph dataset 3DSSG . From left to right: RGB image, rendered depth, rendered dense semantic instance segmentation, dense semantic instance segmentation on textured model, 2D semantic scene graph.

C.3 Statistics

In this paragraph, we present further data statistics. Figure C.3 reports histograms to visualize the statistics related to the number of relationships (a, b) or attributes (c, d) in 3DSSG per 3D scene (scan) as well as per object instance. These statistics show that our scene graphs are quite dense. Figures C.5 and C.6 show the most frequent object, predicate, attribute and affordance occurrences extracted from the 3DSSG graphs. Please note that for visualization simplicity nouns and prepositions are removed from affordances. For example, `hanging in` or `hanging on` are collapsed into `hanging`. Figure C.7 highlight some of the most common semantic connections present in the dataset. Further, Figure C.4 illustrates a few scene examples, where the objects are annotated with the corresponding attributes.



(a) Scenes and number of relationships

(b) Object instances and number of relationships

(c) Object instances and number of attributes

(d) Scenes and number of attributes

Figure C.3. Histograms of number of relationship and attribute occurences in 3DSSG.

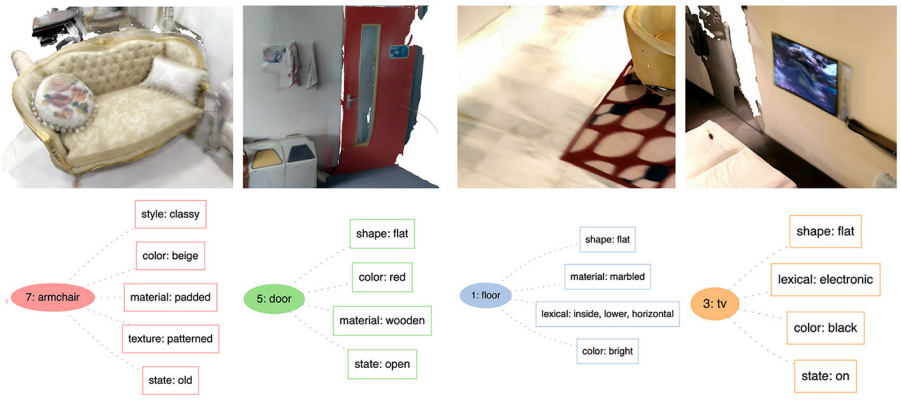
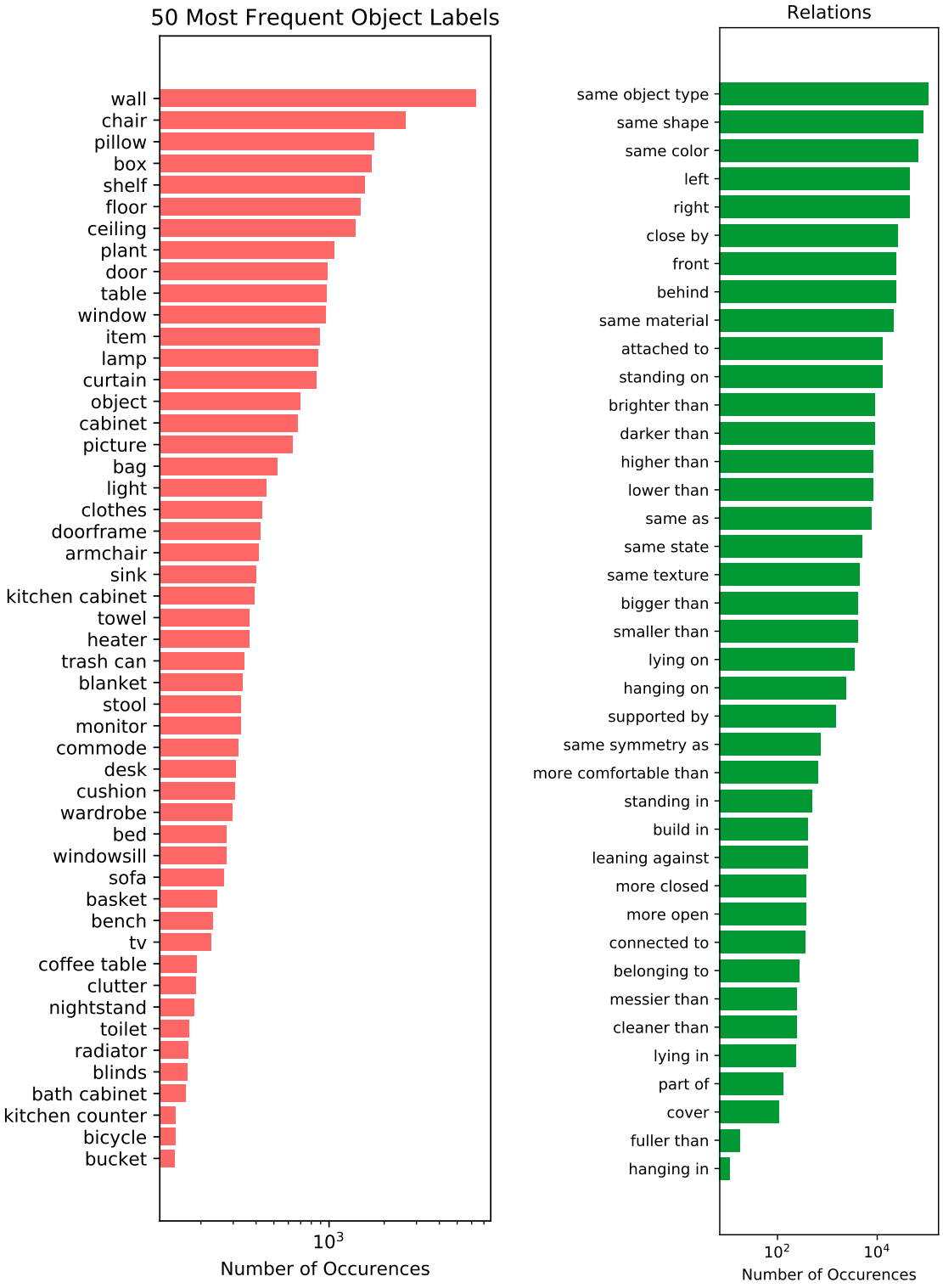


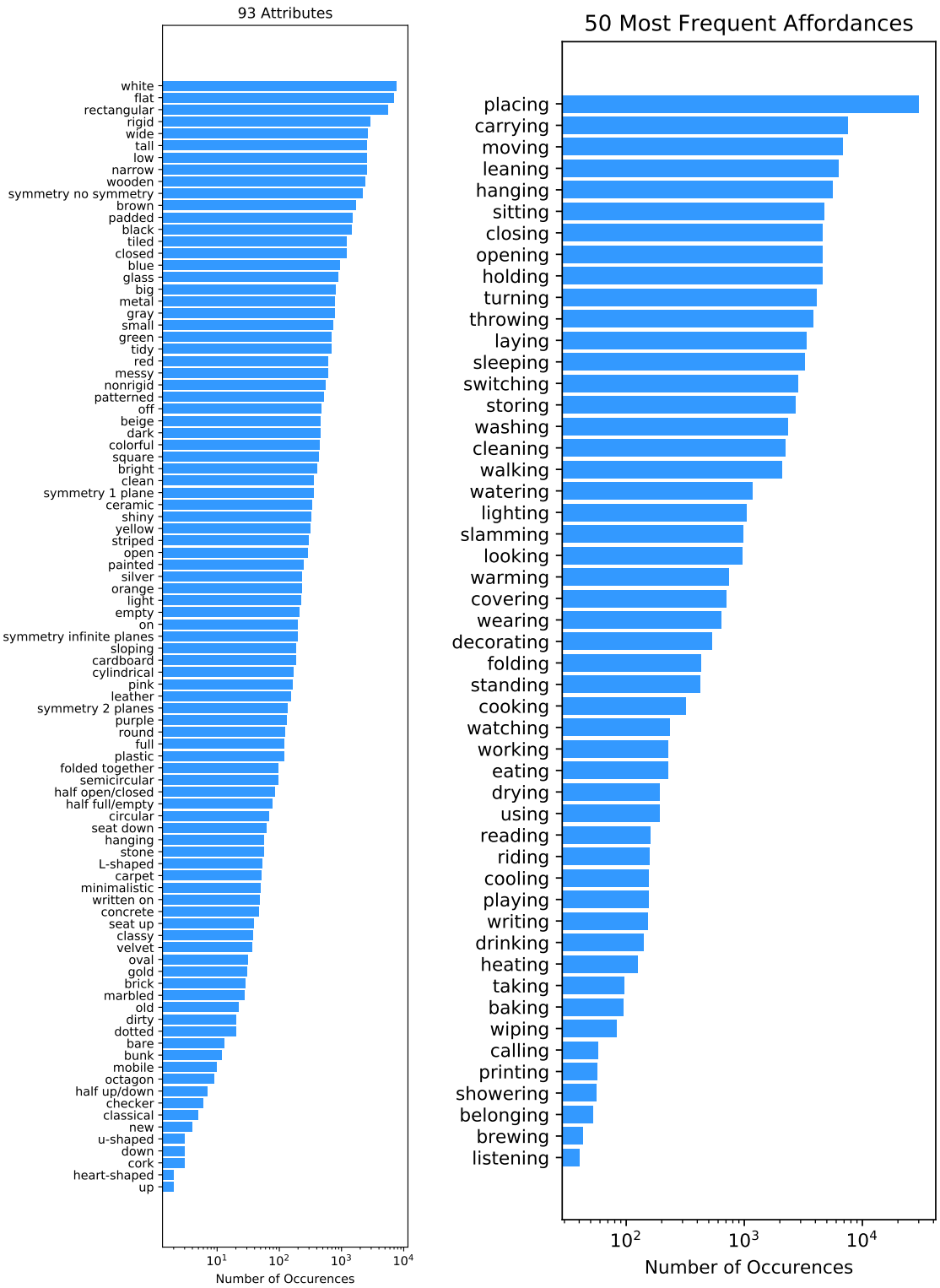
Figure C.4. Example object instances (top) and their corresponding attributes (below).



(a) Most frequent (Top-50) object classes used for the training of the Scene Graph Prediction Network

(b) Predicate classes

Figure C.5. Object and predicate classes, sorted by occurrence, presented in logarithmic scale



(a) Attributes (b) Simplified affordances

Figure C.6. Attributes and affordances in the 3DSSG dataset, sorted by occurrence, presented in logarithmic scale.

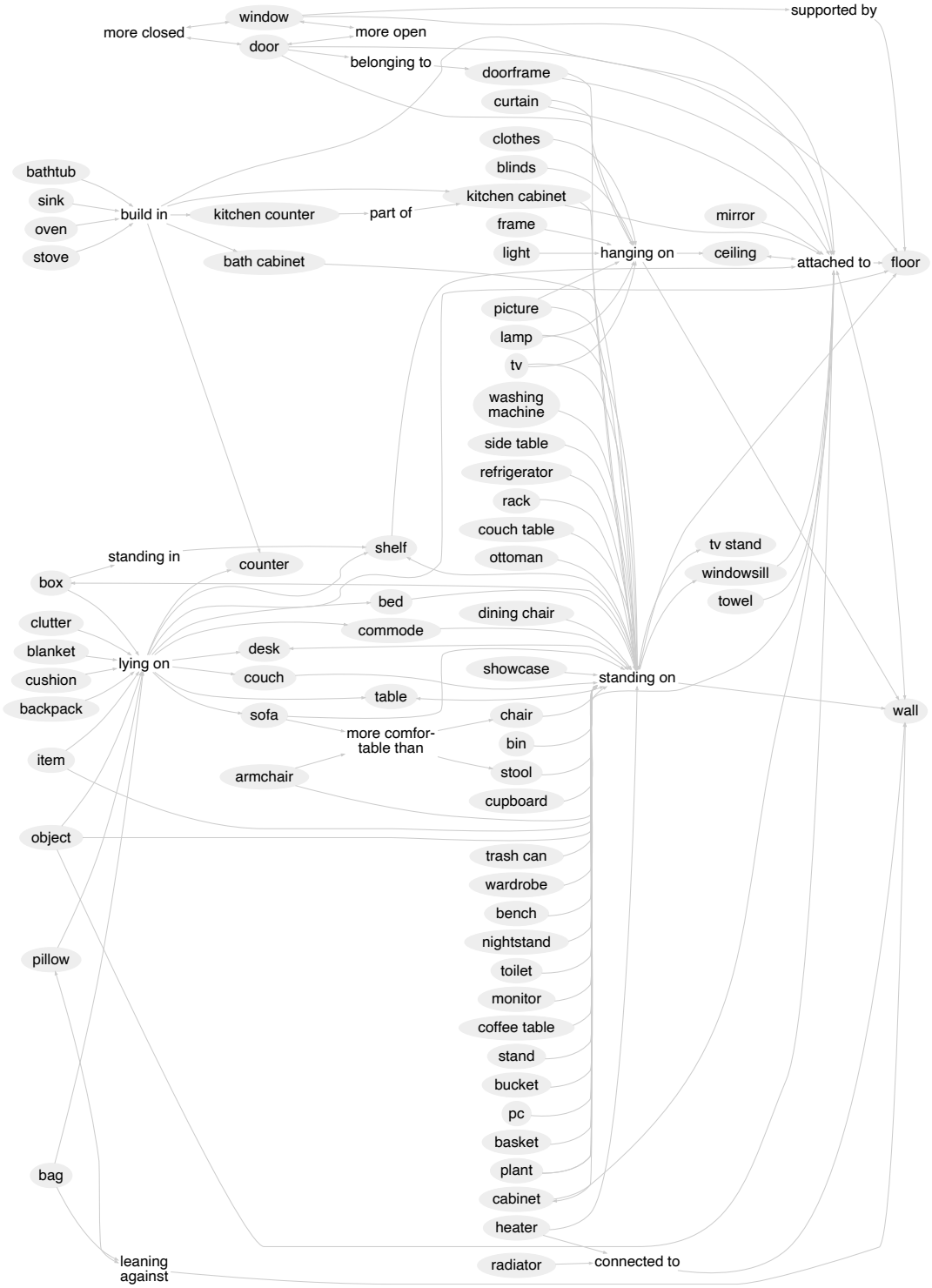


Figure C.7. Most frequent triplets `(subject, predicate, object)` with more than 50 occurrences in 3DSSG. Please note that to simplify visualization, proximity relationships and the majority of comparative relationships are filtered out.

C.4 WordNet graphs

Figure C.8 visualizes the hierarchy of classes obtained via WordNet. By traversing the tree, one can extract the per-node hierarchy of labels c. For an object originally annotated as `bench`, the hierarchical label c would be c = {`bench, seat, furniture, ..., entity`}. Thereby, colored nodes represent class labels from the annotation set, while white nodes are not part of the original label set (typically abstract representations).

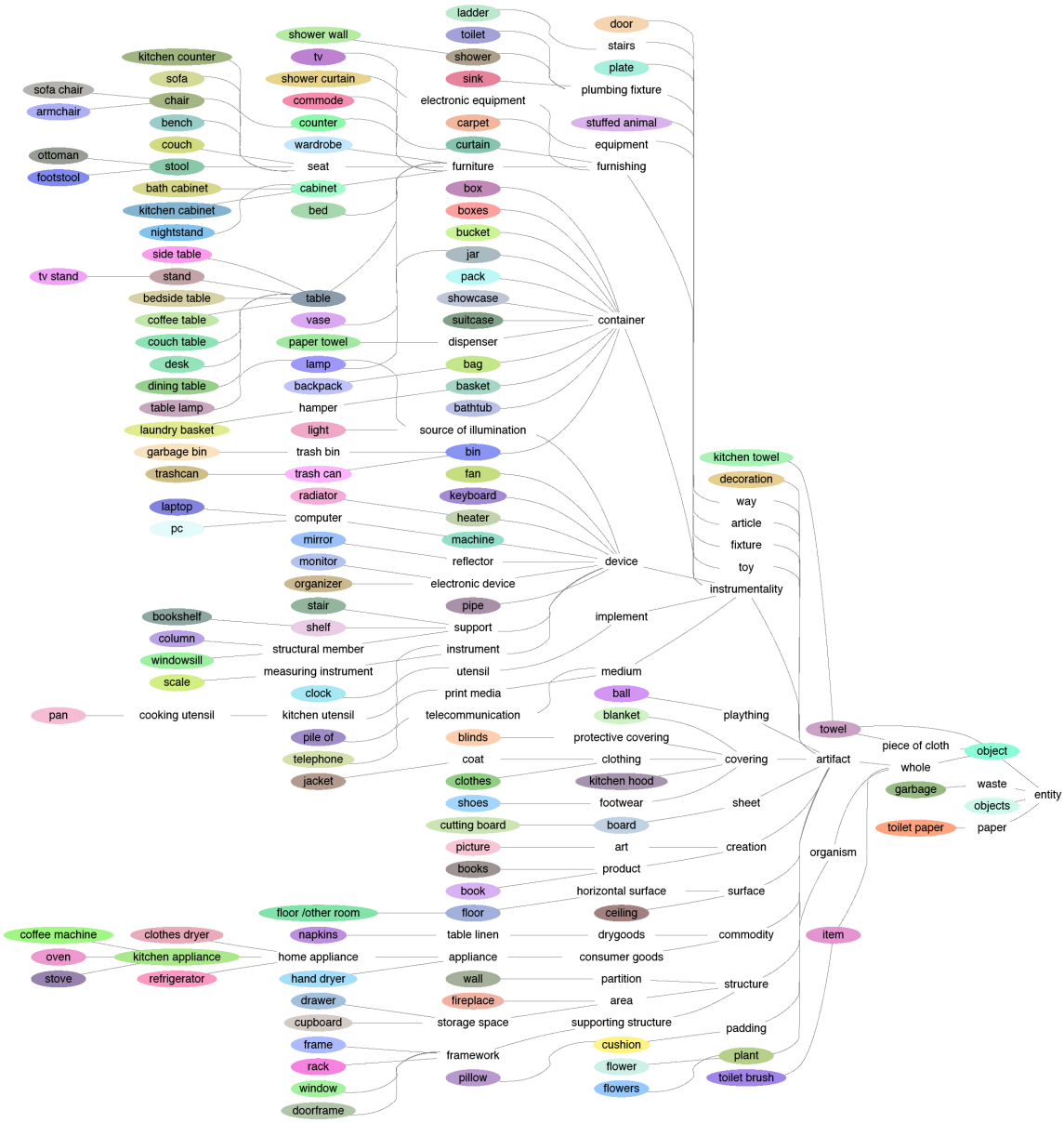


Figure C.8. Visualization of the hierarchical tree of lexical relationships on a (bigger) subset of classes from 3RScan, extending Figure 4.12

Graph-to-3D Details

D

D.1 Discriminator architectures

layer id	layer type	input layer	input channels	output channels
L1	Linear	$(o_i, o_j, r_{ij}, b_i, b_j)$	360	512
L2	Batch Norm	L1	512	512
L3	Leaky-ReLU	L2	512	512
L4	Linear	L3	512	512
L5	Batch Norm	L4	512	512
L6	Leaky-ReLU	L5	512	512
L7	Linear	L6	512	1
out	Sigmoid	L7	1	1

Table D.1. Architecture of D_{box}

layer id	layer type	input layer	input channels	output channels
L1	Linear	e_i^s	128	512
L2	Batch Norm	L1	512	512
L3	Leaky-ReLU	L2	512	512
L4	Linear	L3	512	512
L5	Batch Norm	L4	512	512
L6	Leaky-ReLU	L5	512	512
L7	Linear	L6	512	1
outD	Sigmoid	L7	1	1
L9	Linear	L6	512	160
outC	Softmax	L9	160	160

Table D.2. Architecture of D_{shape}

D.2 Scene graph constraints

As a way to evaluate layout we utilized their fitness with the scene graph constraints. The metrics follow the definitions from Table D.3. We validate all predicate categories in 3DSSG

that can be captured with a geometric rule. Other edges are not straight forward to validate in this manner, as they are manually annotated (*e.g. belonging to*, *leaning against*).

Relationship	Rule
left of	$c_{x,i} < c_{x,j}$ and $\text{iou}(b_i, b_j) < 0.5$
right of	$c_{x,i} > c_{x,j}$ and $\text{iou}(b_i, b_j) < 0.5$
front of	$c_{y,i} < c_{y,j}$ and $\text{iou}(b_i, b_j) < 0.5$
behind of	$c_{y,i} > c_{y,j}$ and $\text{iou}(b_i, b_j) < 0.5$
higher than	$h_i + c_{z,i}/2 > h_j + c_{z,j}/2$
lower than	$h_i + c_{z,i}/2 < h_j + c_{z,j}/2$
smaller than	$w_i l_i h_i < w_j l_j h_j$
bigger than	$w_i l_i h_i > w_j l_j h_j$
same as	$\text{iou}_C(b_i, b_j) > 0.5$

Table D.3. Computation of geometric constraint accuracy, for an instance pair (i, j). iou_C refers to iou computation after both objects have been centered at zero.

D.3 Shape generation networks

Point clouds We utilize AtlasNet [39], to learn a low-dimensional latent embedding on point clouds. AtlasNet – based on PointNet [109] for the encoder part – receives a point cloud and encodes it into a global feature descriptor via $\mathcal{E}_{atlas}$. AtlasNet is particularly suited for the task since the sampling on the uv-map allows to generate point clouds at arbitrarily resolutions while only using a small set of points during training. This leads to very efficient training while saving memory. Further, the resulting point cloud is inferred via the decoder $\mathcal{D}_{atlas}$, using this global feature descriptor together with sampled 2D points from the uv-map. We train AtlasNet on a mixture of synthetic data from ShapeNet and real 3RScan objects, transformed in canonical pose.

Implicit functions Additionally, we utilize DeepSDF [106] to generate shapes represented as implicit functions. For this purpose, we learn class-specific Auto-Decoders trained on the synthetic data from ShapeNet [10]. We use 350 shapes in canonical pose and learn a 128-dimensional continuous shape space. We then label each object in 3RScan with the feature descriptor of the best fitting shape from the training set. Initially, we attempted to use a similar partial scan alignment as originally proposed in [106]. Yet, we empirically found out that it does not work well for 3RScan data, as the point quality is quite low. Thus, we instead utilize the 3D points to find the best matching descriptor from the training set by computing the SDF value for each shape and taking the one with minimal value. As we train our generative model on the basis of these annotations, we can still make use of a continuous shape space.

Bibliography

[1] A. Aguiar and R. Baillargeon. "Developments in young infants' reasoning about occluded objects". In: *Cognitive Psychology* 45 (2002), pp. 267–336 (see p. 6).

[2] I. Armeni, Z.-Y. He, J. Gwak, A. R. Zamir, M. Fischer, J. Malik, and S. Savarese. "3D Scene Graph: A Structure for Unified Semantics, 3D Space, and Camera". In: *International Conference on Computer Vision (ICCV)*. 2019 (see pp. 60, 78–80).

[3] I. Armeni, S. Sax, A. R. Zamir, and S. Savarese. "Joint 2D-3D-Semantic Data for Indoor Scene Understanding". In: *arXiv:1702.01105* (2017) (see pp. 38, 44, 49).

[4] O. Ashual and L. Wolf. "Specifying object attributes and relations in interactive scene generation". In: *ICCV*. 2019, pp. 4561–4569 (see pp. 60, 61, 70, 72, 73, 75).

[5] S. Azadi, D. Pathak, S. Ebrahimi, and T. Darrell. "Compositional gan: Learning conditional image composition". In: *arXiv preprint arXiv:1807.07560* (2018) (see p. 61).

[6] J. Ba, J. R. Kiros, and G. E. Hinton. "Layer Normalization". In: *arXiv:1607.06450* (2016) (see pp. 22, 43).

[7] S. Baker, R. Szeliski, and P. Anandan. "A layered approach to stereo reconstruction". In: *CVPR* (1998) (see p. 29).

[8] J. Z. Bingning Wang Kang Liu. "Conditional Generative Adversarial Networks for Commonsense Machine Comprehension". In: *IJCAI*. 2017 (see p. 24).

[9] A. Chakrabarti, J. Shao, and G. Shakhnarovich. "Depth from a Single Image by Harmonizing Overcomplete Local Network Predictions". In: *NIPS* (2016) (see p. 31).

[10] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012* (2015) (see pp. 88, 132).

[11] Q. Chen and V. Koltun. "Photographic image synthesis with cascaded refinement networks". In: *ICCV*. 2017, pp. 1511–1520 (see pp. 61, 67, 68).

[12] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *arXiv preprint arXiv:1409.1259* (2014) (see p. 105).

[13] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo. "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation". In: *CVPR*. 2018, pp. 8789–8797 (see p. 61).

[14] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction". In: *ECCV*. 2016 (see p. 31).

[15] D. Clevert, T. Unterthiner, and S. Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2016 (see pp. 18, 22).

[16] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes". In: *International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (see pp. 34, 60, 78).

[17] B. Dai, Y. Zhang, and D. Lin. "Detecting visual relationships with deep relational networks". In: *CVPR*. 2017, pp. 3076–3086 (see pp. 60, 68).

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *CVPR* (2009) (see pp. 17, 41, 65).

[19] H. Dhamo, A. Farshad, I. Laina, N. Navab, G. D. Hager, F. Tombari, and C. Rupprecht. "Semantic image manipulation using scene graphs". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5213–5222 (see pp. 1, 7, 9, 11, 23, 60, 61, 70).

[20] H. Dhamo, F. Manhardt, N. Navab, and F. Tombari. "Graph-to-3D: End-to-end Generation and Manipulation of 3D Scenes using Scene Graphs". In: *International Conference on Computer Vision*. 2021 (see pp. 1, 8, 9, 11, 23, 60, 62, 103).

[21] H. Dhamo, N. Navab, and F. Tombari. "Object-Driven Multi-Layer Scene Decomposition From a Single Image". In: *Proceedings IEEE International Conference on Computer Vision (ICCV)*. 2019 (see pp. 1, 7, 9–11, 23, 30, 55, 56, 109).

[22] H. Dhamo, K. Tateno, I. Laina, N. Navab, and F. Tombari. "Peeking behind objects: Layered depth prediction from a single image". In: *Pattern Recognition Letters* 125 (2019), pp. 333–340 (see pp. 1, 7, 9, 10, 23, 30, 31, 36–38, 49, 54–56).

[23] K. Ehsani, R. Mottaghi, and A. Farhadi. "Segan: Segmenting and generating the invisible". In: *CVPR*. 2018 (see pp. 29, 38).

[24] D. Eigen and R. Fergus. "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture". In: *ICCV*. 2015 (see p. 31).

[25] D. Eigen, C. Puhrsch, and R. Fergus. "Depth Map Prediction from a Single Image Using a Multi-scale Deep Network". In: *NIPS*. 2014 (see p. 31).

[26] F. Engelmann, T. Kontogianni, A. Hermans, and B. Leibe. "Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds". In: *International Conference on Computer Vision (ICCV) Workshops*. 2017 (see p. 60).

[27] H. Fan, H. Su, and L. J. Guibas. "A Point Set Generation Network for 3D Object Reconstruction From a Single Image". In: *CVPR*. 2017 (see pp. 31, 38).

[28] C. Fellbaum, ed. *WordNet: an electronic lexical database*. MIT Press, 1998 (see p. 79).

[29] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395 (see p. 88).

[30] M. Fisher, M. Savva, and P. Hanrahan. "Characterizing structural relationships in scenes using graph kernels". In: *ACM Trans. Graph* (2011) (see p. 60).

[31] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. "Deepstereo: Learning to predict new views from the world's imagery". In: *CVPR*. 2016 (see pp. 30, 54).

[32] R. Garg, V. K. BG, G. Carneiro, and I. Reid. "Unsupervised cnn for single view depth estimation: Geometry to the rescue". In: *European Conference on Computer Vision*. Springer. 2016, pp. 740–756 (see pp. 30, 54).

[33] S. Garg, H. Dhamo, A. Farshad, S. Musatian, N. Navab, and F. Tombari. "Unconditional Scene Graph Generation". In: *International Conference on Computer Vision*. 2021 (see pp. 1, 10, 23, 103–105).

[34] S. Ghosh, G. Burachas, A. Ray, and A. Ziskind. "Generating Natural Language Explanations for Visual Question Answering using Scene Graphs and Visual Attention". In: *ArXiv* abs/1902.05715 (2019) (see p. 60).

[35] J. J. Gibson. *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin, 1979 (see p. 80).

[36] C. Godard, O. Mac Aodha, and G. J. Brostow. "Unsupervised monocular depth estimation with left-right consistency". In: *CVPR*. 2017, pp. 270–279 (see pp. 30, 54).

[37] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016 (see p. 16).

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680 (see pp. 6, 24, 41, 91).

[39] T. Groueix, M. Fisher, V. G. Kim, B. Russell, and M. Aubry. "AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation". In: *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2018 (see pp. 89, 95, 98, 132).

[40] R. Guo, C. Zou, and D. Hoiem. "Predicting Complete 3D Models of Indoor Scenes". In: *arXiv:1504.02437* (2015) (see p. 31).

[41] R. Hartley and A Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004 (see p. 13).

[42] K. He, G. Gkioxari, P. Dollár, and R. Girshick. "Mask r-cnn". In: *ICCV*. 2017 (see pp. 39, 40, 42, 49, 109).

[43] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *CVPR* (2016) (see pp. 17, 22, 31, 42).

[44] K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1026–1034 (see pp. 18, 22).

[45] P. Hedman, S. Alsisan, R. Szeliski, and J. Kopf. "Casual 3D Photography". In: (2017) (see pp. 6, 11, 29, 31, 55).

[46] R. Herzig, M. Raboh, G. Chechik, J. Berant, and A. Globerson. "Mapping images to scene graphs with permutation-invariant structured prediction". In: *NeurIPS*. 2018 (see p. 60).

[47] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". In: *NeurIPS*. 2017 (see p. 69).

[48] G. E. Hinton, S. Osindero, and Y.-W. Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554 (see p. 17).

[49] G. E. Hinton et al. "Learning distributed representations of concepts". In: *Proceedings of the eighth annual conference of the cognitive science society*. Vol. 1. Amherst, MA. 1986, p. 12 (see p. 17).

[50] S. Hong, D. Yang, J. Choi, and H. Lee. "Inferring semantic layout for hierarchical text-to-image synthesis". In: *CVPR*. 2018 (see p. 61).

[51] J. Hou, A. Dai, and M. Nießner. "3D-SIS: 3D Semantic Instance Segmentation of RGB-D Scans". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (see p. 60).

[52] S.-M. Hu, F.-L. Zhang, M. Wang, R. R. Martin, and J. Wang. "PatchNet: A Patch-based Image Representation for Interactive Library-driven Image Editing". In: *ACM Trans. Graph.* 32 (2013) (see p. 61).

[53] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. "Densely connected convolutional networks". In: *CVPR*. 2017 (see p. 31).

[54] S. Huang, S. Qi, Y. Xiao, Y. Zhu, Y. N. Wu, and S.-C. Zhu. "Cooperative Holistic Scene Understanding: Unifying 3D Object, Layout, and Camera Pose Estimation". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2018 (see p. 60).

[55] S. Iizuka, E. Simo-Serra, and H. Ishikawa. "Globally and locally consistent image completion". In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), p. 107 (see pp. 24, 35).

[56] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015) (see pp. 22, 68).

[57] P. Isola and C. Liu. "Scene Collaging: Analysis and Synthesis of Natural Images with Semantic Layers". In: *ICCV* (2013) (see pp. 29, 60).

[58] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks". In: *In CVPR* (2017) (see pp. 23, 25, 35, 36, 43, 61).

[59] S. Jae Hwang, S. N. Ravi, Z. Tao, H. J. Kim, M. D. Collins, and V. Singh. "Tensorize, factorize and regularize: Robust visual relationship learning". In: *CVPR*. 2018, pp. 1014–1023 (see p. 60).

[60] C. Jiang, S. Qi, Y. Zhu, S. Huang, J. Lin, L. Yu, D. Terzopoulos, and S. Zhu. "Configurable 3D Scene Synthesis and 2D Image Rendering with Per-pixel Ground Truth Using Stochastic Grammars". In: *International Journal of Computer Vision (IJCV)* (2018) (see p. 61).

[61] J. Jiao, Y. Cao, Y. Song, and R. W. H. Lau. "Look Deeper into Depth: Monocular Depth Estimation with Semantic Booster and Attention-Driven Loss". In: *ECCV*. 2018 (see p. 31).

[62] Y. Jo and J. Park. "SC-FEGAN: Face Editing Generative Adversarial Network with User's Sketch and Color". In: *arXiv preprint arXiv:1902.06838* (2019) (see p. 61).

[63] J. Johnson, R. Krishna, M. Stark, L. Li, D. A. Shamma, M. S. Bernstein, and L. Fei-Fei. "Image retrieval using scene graphs". In: *CVPR*. 2015 (see p. 60).

[64] J. Johnson, A. Alahi, and L. Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". In: *ECCV*. 2016 (see p. 41).

[65] J. Johnson, A. Gupta, and L. Fei-Fei. "Image Generation from Scene Graphs". In: *CVPR*. 2018 (see pp. 7, 60–62, 67–70, 72, 73, 103, 105, 106).

[66] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C Lawrence Zitnick, and R. Girshick. "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning". In: *CVPR*. 2017 (see pp. 69, 70).

[67] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. "Analyzing and improving the image quality of stylegan". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8110–8119 (see pp. 106, 110).

[68] K. Karsch, C. Liu, and S. B. Kang. "Depth Transfer: Depth Extraction from Videos Using Non-parametric Sampling". In: *DICCV*. 2016 (see p. 31).

[69] A. Kendall and Y. Gal. "What uncertainties do we need in bayesian deep learning for computer vision?" In: *NIPS*. 2017, pp. 5580–5590 (see p. 31).

[70] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014) (see pp. 19, 43, 69).

[71] D. P. Kingma and M. Welling. "Auto-encoding variational bayes". In: *ICLR* (2013) (see pp. 6, 24).

[72] J. Konrad, M. Wang, and P. Ishwar. "2d-to-3d image conversion by learning depth from examples". In: *CVPR Workshops*. 2012 (see p. 31).

[73] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations". In: *IJCV* 123.1 (2017), pp. 32–73 (see pp. 69, 72, 78, 124).

[74] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (see pp. 17, 22).

[75] N. Kulkarni, I. Misra, S. Tulsiani, and A. Gupta. "3D-RelNet: Joint Object and Relational Network for 3D Prediction". In: *International Conference on Computer Vision (ICCV)* (2019) (see p. 60).

[76] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. "Deeper depth prediction with fully convolutional residual networks". In: *2016 Fourth international conference on 3D vision (3DV)*. IEEE. 2016, pp. 239–248 (see pp. 31, 34, 41–43, 49, 109).

[77] G. Lample, N. Zeghidour, N. Usunier, A. Bordes, L. Denoyer, et al. "Fader networks: Manipulating images by sliding attributes". In: *NeurIPS*. 2017, pp. 5967–5976 (see p. 61).

[78] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (see p. 21).

[79] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. "Photo-realistic single image super-resolution using a generative adversarial network". In: *CVPR*. 2017, pp. 4681–4690 (see pp. 25, 61).

[80] B. Li, C. Shen, Y. Dai, A. van den Hengel, and M. He. "Depth and surface normal estimation from monocular images using regression on deep features and hierarchical CRFs". In: *CVPR*. 2015 (see p. 31).

[81] C. Li and M. Wand. "Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks". In: *ECCV*. Ed. by B. Leibe, J. Matas, N. Sebe, and M. Welling. 2016 (see p. 25).

[82] G. Li, M. Müller, A. Thabet, and B. Ghanem. "DeepGCNs: Can GCNs Go as Deep as CNNs?" In: *International Conference on Computer Vision (ICCV)*. 2019 (see p. 63).

[83] M. Li, A. Gadi Patil, K. Xu, S. Chaudhuri, O. Khan, A. Shamir, C. Tu, B. Chen, D. Cohen-Or, and H. Zhang. "GRAINS: Generative Recursive Autoencoders for Indoor Scenes". In: *ACM Transactions on Graphics (TOG)* (2018) (see pp. 60, 61).

[84] Y. Li, W. Ouyang, B. Zhou, J. Shi, C. Zhang, and X. Wang. "Factorizable net: an efficient subgraph-based framework for scene graph generation". In: *ECCV*. 2018, pp. 335–351 (see pp. 60, 64, 65, 68).

[85] Y. Li, W. Ouyang, B. Zhou, K. Wang, and X. Wang. "Scene graph generation from objects, phrases and region captions". In: *ICCV*. 2017, pp. 1261–1270 (see p. 60).

[86] Y. Li, Z. Gan, Y. Shen, J. Liu, Y. Cheng, Y. Wu, L. Carin, D. Carlson, and J. Gao. "StoryGAN: A Sequential Conditional GAN for Story Visualization". In: *arXiv preprint arXiv:1812.02784* (2018) (see p. 61).

[87] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. "Focal Loss for Dense Object Detection". In: *International Conference on Computer Vision (ICCV)*. 2017 (see p. 83).

[88] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755 (see p. 49).

[89] B. Liu, S. Gould, and D. Koller. "Single image depth estimation from predicted semantic labels". In: *CVPR* (2010) (see p. 31).

[90] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz. "PlaneRCNN: 3D Plane Detection and Reconstruction From a Single Image". In: *CVPR*. 2019 (see p. 29).

[91] C. Liu, P. Kohli, and Y. Fukurawa. "Layered scene decomposition via the occlusion-crf". In: *CVPR*. 2016 (see p. 29).

[92] F. Liu, C. Shen, and G. Lin. "Deep Convolutional Neural Fields for Depth Estimation from a Single Image". In: *CVPR* (2015) (see p. 31).

[93] M. Liu, M. Salzmann, and X. He. "Discrete-Continuous Depth Estimation from a Single Image". In: *CVPR*. 2014, pp. 716–723 (see p. 31).

[94] T. Liu, S. Chaudhuri, V. Kim, Q. Huang, N. Mitra, and T. Funkhouser. "Creating Consistent Scene Graphs Using a Probabilistic Grammar". In: *ACM Transactions on Graphics (TOG)* (2014) (see p. 60).

[95] C. Lu, R. Krishna, M. Bernstein, and L. Fei-Fei. "Visual Relationship Detection with Language Priors". In: *European Conference on Computer Vision (ECCV)*. 2016 (see pp. 60, 82, 84, 85).

[96] A. Luo, Z. Zhang, J. Wu, and J. B. Tenenbaum. "End-to-End Optimization of Scene Layout". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (see pp. 8, 60, 62, 87, 93–95, 97, 98).

[97] R. Ma, A. G. Patil, M. Fisher, M. Li, S. Pirk, B.-S. Hua, S.-K. Yeung, X. Tong, L. Guibas, and H. Zhang. "Language-Driven Synthesis of 3D Scenes from Scene Databases". In: *SIGGRAPH Asia, Technical Papers*. 2018 (see p. 61).

[98] J. McCormac, A. Handa, S. Leutenegger, and A. J.Davison. "SceneNet RGB-D: 5M Photorealistic Images of Synthetic Indoor Trajectories with Ground Truth". In: *ICCV* (2017) (see pp. 33, 44, 46, 47).

[99] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (see p. 16).

[100] M. Mirza and S. Osindero. "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014) (see p. 61).

[101] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. Mitra, and L. Guibas. "StructureNet: Hierarchical Graph Networks for 3D Shape Generation". In: *ACM Transactions on Graphics (TOG)* (2019) (see pp. 61, 111).

[102] P. K. Nathan Silberman Derek Hoiem and R. Fergus. "Indoor Segmentation and Support Inference from RGBD Images". In: *ECCV*. 2012 (see pp. 44, 47, 48, 80).

[103] A. Newell and J. Deng. "Pixels to graphs by associative embedding". In: *NeurIPS*. 2017, pp. 2171–2180 (see pp. 60, 64).

[104] Y. Nie, X. Han, S. Guo, Y. Zheng, J. Chang, and J. J. Zhang. "Total3DUnderstanding: Joint Layout, Object Pose and Mesh Reconstruction for Indoor Scenes From a Single Image". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (see p. 61).

[105] A. Odena, C. Olah, and J. Shlens. "Conditional image synthesis with auxiliary classifier GANs". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 2017, pp. 2642–2651 (see pp. 61, 67, 91).

[106] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. "Deepsdf: Learning continuous signed distance functions for shape representation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 165–174 (see pp. 89, 95, 98, 132).

[107] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu. "Semantic Image Synthesis with Spatially-Adaptive Normalization". In: *CVPR*. 2019 (see pp. 67, 68, 109).

[108] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. "Context Encoders: Feature Learning by Inpainting". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (see pp. 24, 35, 61).

[109] C. R. Qi, H. Su, K. Mo, and L. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (see pp. 60, 82, 132).

[110] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas. "Volumetric and Multi-View CNNs for Object Classification on 3D Data". In: *Computer Vision and Pattern Recognition (CVPR)*. 2016 (see p. 60).

[111] C. R. Qi, L. Yi, H. Su, and L. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2017 (see p. 60).

[112] M. Qi, W. Li, Z. Yang, Y. Wang, and J. Luo. "Attentive Relational Networks for Mapping Images to Scene Graphs". In: *arXiv preprint arXiv:1811.10696* (2018) (see p. 60).

[113] N. Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151 (see p. 19).

[114] M. Ramamonjisoa, Y. Du, and V. Lepetit. "Predicting Sharp and Accurate Occlusion Boundaries in Monocular Depth Estimation Using Displacement Fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (see p. 31).

[115] M. Ramamonjisoa and V. Lepetit. "SharpNet: Fast and Accurate Recovery of Occluding Contours in Monocular Depth Estimation". In: *The IEEE International Conference on Computer Vision (ICCV) Workshops* (2019) (see p. 31).

[116] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. "Generative adversarial text to image synthesis". In: *arXiv preprint arXiv:1605.05396* (2016) (see p. 61).

[117] S. Ren, K. He, R. Girshick, and J. Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99 (see p. 60).

[118] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *MICCAI* (2015) (see pp. 23, 41, 43).

[119] F. Rosenblatt. "The Perceptron: {A} Probabilistic Model for Information Storage and Organization in the Brain". In: *Psychological Review* 65 (1958), pp. 386–408 (see p. 16).

[120] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536 (see p. 17).

[121] M. A. Sadeghi and A. Farhadi. "Recognition using visual phrases". In: *CVPR*. 2011 (see p. 60).

[122] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "Improved Techniques for Training GANs". In: *NeurIPS*. 2016 (see p. 69).

[123] A. Saxena, S. H. Chung, and A. Y. Ng. "Learning depth from single monocular images". In: *NIPS*. 2006 (see p. 31).

[124] A. Saxena, M. Sun, and A. Y. Ng. "Make3d: Learning 3d scene structure from a single still image". In: *PAMI* (2009) (see p. 31).

[125] J. Shade, S. Gortler, L.-w. He, and R. Szeliski. "Layered Depth Images". In: *SIGGRAPH '98*. 1998, pp. 231–242 (see pp. 6, 14, 15, 29, 44).

[126] R. R. Shetty, M. Fritz, and B. Schiele. "Adversarial scene editing: Automatic object removal from weak supervision". In: *NeurIPS*. 2018, pp. 7717–7727 (see p. 61).

[127] Y. Shi, A. X. Chang, Z. Wu, M. Savva, and K. Xu. "Hierarchy Denoising Recursive Autoencoders for 3D Scene Layout Prediction". In: *Computer Vision and Pattern Recognition (CVPR)*. 2019 (see p. 60).

[128] M.-L. Shih, S.-Y. Su, J. Kopf, and J.-B. Huang. "3D Photography using Context-aware Layered Depth Inpainting". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (see pp. 30, 57, 58).

[129] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv:1409.1556* (2014) (see pp. 41, 65, 68).

[130] S. Song, S. Lichtenberg, and J. Xiao. "SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (see p. 60).

[131] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. "Semantic Scene Completion from a Single Depth Image". In: *CVPR* (2017) (see pp. 31, 38, 44).

[132] E. S. Spelke. "Principles of object perception". In: *Cognitive science* 14.1 (1990), pp. 29–56 (see p. 6).

[133] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. "Multi-view Convolutional Neural Networks for 3D Shape Recognition". In: *International Conference on Computer Vision (ICCV)*. 2015 (see p. 60).

[134] D. Sun, E. B. Sudderth, and M. J. Black. "Layered segmentation and optical flow estimation over time". In: *CVPR* (2012) (see p. 29).

[135] W. Sun and T. Wu. "Image Synthesis From Reconfigurable Layout and Style". In: *ICCV*. 2019 (see p. 61).

[136] J. Tighe, M. Niethammer, and S. Lazebnik. "Scene Parsing with Object Instances and Occlusion Ordering". In: *CVPR* (2014) (see p. 29).

[137] R. Tucker and N. Snavely. "Single-view View Synthesis with Multiplane Images". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (see pp. 30, 57).

[138] S. Tulsiani, S. Gupta, D. Fouhey, A. A. Efros, and J. Malik. "Factoring Shape, Pose, and Layout from the 2D Image of a 3D Scene". In: *CVPR*. 2018 (see pp. 31, 61).

[139] S. Tulsiani, R. Tucker, and N. Snavely. "Layer-structured 3D Scene Inference via View Synthesis". In: *ECCV*. 2018 (see pp. 7, 9, 30, 38, 44, 45, 48–51, 54, 55, 57).

[140] J. Wald, A. Avetisyan, N. Navab, F. Tombari, and M. Nießner. "Rio: 3d object instance re-localization in changing indoor environments". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 7658–7667 (see pp. 8, 60, 78, 84).

[141] J. Wald, H. Dhamo, N. Navab, and F. Tombari. "Learning 3D Semantic Scene Graphs from 3D Indoor Reconstructions". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (see pp. 1, 8, 9, 11, 59, 60, 93, 98, 110).

[142] J. Y. A. Wang and E. H. Adelson. "Representing moving images with layers". In: *IEEE transactions on image processing* (1994) (see p. 29).

[143] K. Wang, Y. Lin, B. Weissmann, M. Savva, A. X. Chang, and D. Ritchie. "PlanIT: planning and instantiating indoor scenes with relation graph and spatial prior networks". In: *ACM Trans. Graph.* 38 (2019), 132:1–132:15 (see pp. 62, 94).

[144] K. Wang, M. Savva, A. X. Chang, and D. Ritchie. "Deep convolutional priors for indoor scene synthesis". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), p. 70 (see p. 62).

[145] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. L. Yuille. "Towards Unified Depth and Semantic Prediction From a Single Image". In: *CVPR*. 2015 (see p. 31).

[146] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs". In: *CVPR*. 2018 (see pp. 7, 109).

[147] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. "High-resolution image synthesis and semantic manipulation with conditional gans". In: *CVPR*. 2018, pp. 8798–8807 (see pp. 61, 93).

[148] Y. Wang, D. J. Tan, N. Navab, and F. Tombari. "Adversarial Semantic Scene Completion from a Single Depth Image". In: *3DV* (2018) (see p. 31).

[149] O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson. "SynSin: End-to-End View Synthesis From a Single Image". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (see p. 30).

[150] J. Wu, Y. Wang, T. Xue, X. Sun, W. T. Freeman, and J. B. Tenenbaum. "MarrNet: 3D Shape Reconstruction via 2.5D Sketches". In: *NIPS*. 2017 (see p. 31).

[151] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. "Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling". In: *Advances In Neural Information Processing Systems*. 2016, pp. 82–90 (see pp. 25, 31).

[152] Y. Wu and K. He. "Group normalization". In: *ECCV*. 2018, pp. 3–19 (see p. 22).

[153] F. Xia, A. R. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese. "Gibson Env: Real-World Perception for Embodied Agents". In: *Computer Vision and Pattern Recognition (CVPR)*. 2018 (see pp. 78, 80).

[154] J. Xie, R. Girshick, and A. Farhadi. "Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks". In: *ECCV*. 2016 (see pp. 30, 54).

[155] D. Xu, E. Ricci, W. Ouyang, X. Wang, and N. Sebe. "Multi-scale Continuous CRFs as Sequential Deep Networks for Monocular Depth Estimation". In: *CVPR*. 2017 (see p. 31).

[156] D. Xu, Y. Zhu, C. Choy, and L. Fei-Fei. "Scene Graph Generation by Iterative Message Passing". In: *CVPR*. 2017 (see pp. 60, 64, 82, 85).

[157] X. Yan, J. Yang, K. Sohn, and H. Lee. "Attribute2image: Conditional image generation from visual attributes". In: *ECCV*. 2016, pp. 776–791 (see p. 61).

[158] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh. "Graph r-cnn for scene graph generation". In: *ECCV*. 2018, pp. 670–685 (see pp. 60, 82, 85).

[159] Y. Yang, S. Hallman, D. Ramanan, and C. C. Fowlkes. "Layered Object Models for Image Segmentation". In: *PAMI* (2012) (see p. 29).

[160] S. Yao, T. M. Hsu, J.-Y. Zhu, J. Wu, A. Torralba, B. Freeman, and J. Tenenbaum. "3D-aware scene manipulation via inverse graphics". In: *NeurIPS*. 2018 (see p. 61).

[161] R. A. Yeh, C. Chen, T. Yian Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do. "Semantic image inpainting with deep generative models". In: *CVPR*. 2017, pp. 5485–5493 (see p. 61).

[162] G. Yin, L. Sheng, B. Liu, N. Yu, X. Wang, J. Shao, and C. Change Loy. "Zoom-net: Mining deep feature interactions for visual relationship recognition". In: *ECCV*. 2018, pp. 322–338 (see p. 60).

[163] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models". In: *ICML* (2018) (see pp. 103–105).

[164] M. D. Zeiler. "Adadelta: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012) (see p. 19).

[165] R. Zellers, M. Yatskar, S. Thomson, and Y. Choi. "Neural motifs: Scene graph parsing with global context". In: *CVPR*. 2018, pp. 5831–5840 (see pp. 60, 64).

[166] F. Zhan, H. Zhu, and S. Lu. "Spatial Fusion GAN for Image Synthesis". In: *arXiv preprint arXiv:1812.05840* (2018) (see p. 61).

[167] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas. "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks". In: *ICCV*. 2017, pp. 5907–5915 (see p. 61).

[168] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". In: *CVPR*. 2018 (see p. 69).

[169] Y. Zhang and T. Funkhouser. "Deep Depth Completion of a Single RGB-D Image". In: *CVPR* (2018) (see p. 41).

[170] Z. Zhang, Y. Xie, and L. Yang. "Photographic text-to-image synthesis with a hierarchically-nested adversarial network". In: *CVPR*. 2018 (see p. 61).

[171] B. Zhao, B. Chang, Z. Jie, and L. Sigal. "Modular generative adversarial networks". In: *ECCV*. 2018 (see p. 61).

[172] B. Zhao, L. Meng, W. Yin, and L. Sigal. "Image Generation from Layout". In: (2019) (see p. 61).

[173] Y. Zhao and S. chun Zhu. "Image Parsing with Stochastic Scene Grammar". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2011 (see p. 60).

[174] Y. Zhao, B. L. Price, S. Cohen, and D. Gurari. "Guided Image Inpainting: Replacing an Image Region by Pulling Content From Another Image". In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2018), pp. 1514–1523 (see p. 61).

[175] Y. Zhao, T. Birdal, H. Deng, and F. Tombari. "3D Point Capsule Networks". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (see p. 60).

[176] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. "Unsupervised learning of depth and ego-motion from video". In: *CVPR*. 2017 (see pp. 30, 54).

[177] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely. "Stereo Magnification: Learning View Synthesis Using Multiplane Images". In: *ACM Trans. Graph.* (2018) (see pp. 6, 16, 30).

[178] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros. "View synthesis by appearance flow". In: *ECCV*. 2016 (see pp. 30, 44, 54, 55).

[179] Y. Zhou, Z. While, and E. Kalogerakis. "SceneGraphNet: Neural Message Passing for 3D Indoor Scene Augmentation". In: *IEEE Conference on Computer Vision (ICCV)*. 2019 (see p. 60).

[180] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros. "Generative visual manipulation on the natural image manifold". In: *ECCV*. 2016, pp. 597–613 (see p. 24).

[181] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. "Unpaired image-to-image translation using cycle-consistent adversarial networks". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232 (see pp. 25, 61).

[182] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. "High-quality Video View Interpolation Using a Layered Representation". In: *ACM SIGGRAPH*. 2004 (see p. 29).

[183] C. Zou, A. Colburn, Q. Shan, and D. Hoiem. "LayoutNet: Reconstructing the 3D Room Layout from a Single RGB Image". In: *arXiv:1803.08999* (2018) (see p. 31).

List of Figures

List of Tables