Institut für Robotik und Mechatronik	BJ.: 2021
	IB.Nr.: DLR-IB-RM-OP-2021-225

MASTERARBEIT

INDOOR SCENE SYNTHESIS: GENERATION OF INTERIOR DESIGNS WITH TRANSFORMER MODELS

Freigabe:	Der Bearbeiter:	Unterschriften
	Mohamad Elbadrawy	mohamad elbadrawy
	Betreuer:	
	Maximilian Denninger	
	Der Institutsdirektor	
	Prof. Alin Albu-Schäffer	

Dieser Bericht enthält 60 Seiten, 34 Abbildungen und 8 Tabellen

Ort: Oberpfaffenhofen	Datum: 06.12.2021	Bearbeiter: Maximilian Denninger	Zeichen:

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Indoor Scene Synthesis: Generation of Interior Designs with Transformer Models

Mohamad Elbadrawy

DEPARTMENT OF INFORMATICS

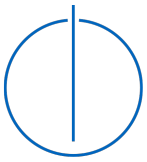
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Indoor Scene Synthesis: Generation of Interior Designs with Transformer Models

Indoor Scene Synthesis: Generierung von Interior Designs mit Transformatormodellen

Author:	Mohamad Elbadrawy
Supervisor:	Dr. habil. Rudolph Triebel
Advisor:	Denninger Maximilian
Submission Date:	15th September 2021



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15th September 2021 Mohamad Elbadrawy

mohamad elbadrawy

Acknowledgments

Firstly I would like to thank the DLR and Dr. Rudolph Triebel, for giving me the opportunity to work on my Master Thesis and expand my practical experience in Deep Learning and Computer Vision. I would also like to thank my supervisor Maximillian Denninger, who has been very supportive throughout my time at DLR. Additionally, I would like to thank the rest of my team at DLR for helping me write my thesis.

Abstract - English

The task of indoor scene generation consists of creating a sequence of objects categories, their locations, and orientations conditioned on the shape and size of a room. In order to automize the easy and fast generation of authentic 3D scenes without the need of human supervision. For that large scale indoor scene data-sets are used, which allow us to extract patterns from expert-designed indoor scenes. Based on those it is possible to generate new scenes based on the learned patterns hidden in these datasets.

Existing methods rely on representing the scene by its 2D or 3D appearance [SZW19] [Ma+17]. There there are other methods representing it as a graph while making assumptions about the possible relations between objects. In contrast to those, the aim of this thesis is to learn relations between objects using the self-attention mechanism of transformers. This approach leads to a similar or even better levels of realism, while being more salable.

Our approach named EnvironmentFormer is simple and an effective generative model conditioned on the room shape, and room type. The room type here could be for example be a kitchen or a living room. Our method is built on the transformer model from Vaswani et al. [Vas+17]. The result of our method is a complete 3D scene, containing information where which objects should be placed. Those results are then further improved by using appearance-based methods for refinement and style consistency.

Abstract - German

Die synthetische Erzeugung von Innenraumszenen besteht daraus, eine Abfolge von Objektkategorien, deren Positionen und Ausrichtungen in Abhängigkeit von der Form und Größe eines Raumes zu erstellen. Das Ziel hier ist die einfache und schnelle Generierung authentischer 3D-Szenen zu automatisieren, ohne dass ein menschlicher Eingriff erforderlich ist. Dazu Werden große Datensätze von Innenraumszenen verwendet, sterben es uns erlauben, Muster aus den von Experten entworfenen Innenraumszenen zu extrahieren. Auf diesen Aufbaud ist es möglich, neue Szenen zu generieren, sterben auf den gelernten Mustern basieren, welche in den Datensätzen versteckt sind.

Be Methoden basierend auf der Darstellung der Szene durch 2D- oder 3D-Darstellung [SZW19] [Ma+17]. Es gibt zusätzliche andere Methoden, sterben sterben Szene als Graph darstellen und dabei Annahmen über sterben Beziehungen zwischen verschiedenen Objekten treffen. Im Gegensatz dazu ist es das Ziel dieser Arbeit, Beziehungen zwischen Objekten zu lernen, wie man den Selbstbeobachtungsmechanismus von Transformern nutzt.

Unser Ansatz namens EnvironmentFormer ist ein einfaches und effektives generatives Modell, das auf der Raumform basiert und dem Raumtyp Szenen generiert. Der Raumtyp könnte hier zum Beispiel eine Küche oder ein Wohnzimmer sein. Unsere Methode dabei baut auf dem Transformermodell von Vaswani et .al [Vas+17] auf. Als Ergebnis liefert unserer Methode eine vollständige 3D-Szene, sterben Informationen darüber enthält, wo welche Objekte platziert werden sollen. This Ergebnisse Werden Dann Mit Hilfe Von Stilbasierten Methoden Zur Verfeinerung Des Stilkonsistenz Weiter Verbessert.

Contents

1 Introduction

People spend a large percentage of their lives indoors -in bedrooms, living rooms, offices, kitchens, and other such spaces- which increases the demand for indoor virtual 3D environments. The usage for those 3D virtual environments has increased between game developers, VR/AR designers, architects, and interior designers, and even people who sometimes just want to sketch their home and view it in 3D. Not only that but AI, vision and robotics researchers had to turn to virtual environments [Son+17] when the real data was no longer enough for the algorithms that kept getting more powerful.

As the vision community turns from passive internet-images-based vision tasks to applications such as the ones listed above, the need for virtual 3D environments becomes critical. The community has recently benefited from large-scale datasets of both synthetic 3D environments and reconstructions of real spaces, and the development of 3D simulation frameworks for studying embodied agents. While these existing datasets are a valuable resource, they are also finite in size and don't adapt to the needs of different vision tasks. To enable large-scale embodied visual learning in 3D environments, one must go beyond such static datasets and instead pursuing the automatic synthesis of novel task-relevant virtual environments.

Due to the rising interest and demand for 3D data modeling, which motivated this work, one decided to tackle the indoor 3D virtual environments sub problem, those environments are used in popular design-your-home 3D tools, tools like Planner5D, Sweet Home 3D, or even Sketch-up, these tools help you design the interior of your home by simply asking you for the floor plan (in most cases) and let you try adding objects and rearranging them, bridging the gap between your vision of your home and what it is actually going to look like by giving you visual feedback of your vision. Also, the increased reliance on 3D modeled environments has dramatically increased across multiple domains, for instance, the size of gaming environments of most games has increased, in some cases, it takes the designing teams years to model such complex game environments.

The goal of this thesis is to investigate the task of automatic synthesis of novel task-relevant virtual environments, specifically indoor environments. This work investigate existing

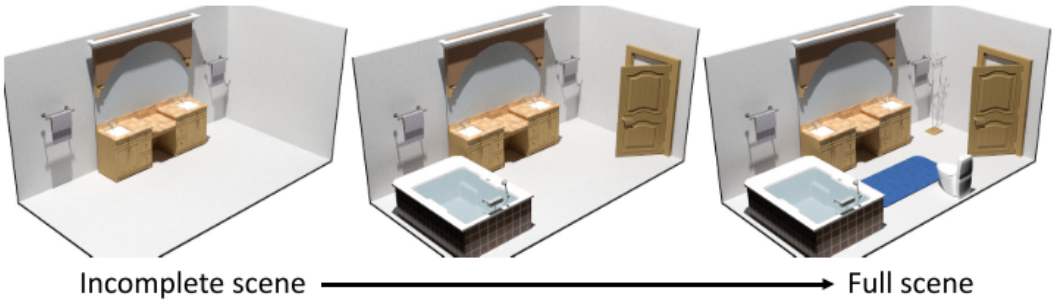


Figure 1.1: A scene completion task done by [ZWK19]. This scene shows a bathroom completed in iterative way.

deep learning based approaches and investigates the application of a transformer model to the task.

1.1 Problem Statement and notation

The problem of automatic synthesis of indoor scenes is the problem of “where” to put “what”. For example, “what” objects are relevant to a bedroom room, a bed is likely to be placed in a bedroom and unlikely to be placed in a kitchen, while a stove is likely to be placed in a kitchen and not in a bedroom. The "where" is more difficult because of the many more possible correct answers to this question. There could be many possible ways to place a bed in a bedroom room. This bed needs to placed where it would be surrounded with bed stands and there is room around it. There should be some enough space to get on the bed from the side. This work answers the question to "what" and "where" in an auto-regressive manner, hence a sequence generation is used to treat the **scene** generation problem as a **sequence** generation problem. Hence opening the way for sequence models to be used to solve it, in this thesis transformer models.

1.2 Thesis Structure

This thesis is structured as follows. In chapter 1, an introduction to virtual 3D scenes generation is provided along with the problem statement of this thesis and the thesis structure.

In chapter 2 an overview of existing approaches to the problem statement is given, highlighting the differences to our approach.

Afterwards, in chapter 3, our approaches to the task of applying transformers networks to scene generation are discussed in detail.

Afterwards, in chapter 4, the experimental setup for the thesis which includes the datasets used, model architectures used, training and evaluation procedures, and a few implementation details are discussed.

Following that, the experiments 5 summarizes our experiments and shows our training graphs. In chapter 6, evaluation of our approaches and extension works are listed and interpreted. Later in chapter 7, future steps and ideas that are out of scope for this thesis but are related to our work are discussed. At the end in chapter 8, our work in this thesis is summarized.

2 Related Work

This chapter provides an overview of existing indoor scene synthesis techniques and states how their approaches differ from this approach.

Scene understanding is about parsing existing scenes into semantic parts. Scenes could be represented by 3D models, point clouds, a graph, RGB-depth scans, or even just RGB images, with each representation having its advantages and uses [Liu+14] [JGS19] [NKP18]. Scene understanding has many uses as 3D reconstruction, object recognition and segmentation [Zho+18]. It is the inverse of scene synthesis which this work tries to solve by generating a scene instead of understanding and analysing an existing one.

To solve the indoor scene synthesis problem, some approaches tried to make use of a graph like structure of an indoor scenes, using models and learning algorithms which are designed to work well on graph structured data as in [ZWK19] where they represent the scene in a graph structure using hand crafted features to form the edges in the graph. To construct such a graph, a heuristic to construct the edges is needed. Some intutive objects relations could be used as such heuristics, typically those heuristics have, Surrounding an object, On top of an object and Between two objects.

By using a neural message passing algorithm they were able to learn those relations between the objects in the scene. And during generation, which is auto-regressive, if the current input object is a bed, and if the predicted next object is a pillow then larger weight would be given for the on-top-of relation. While this work produced some visually good results as seen in figure 1.1, it depends on the quality of the selected heuristics that the graph is built on.

Our work is different than [ZWK19] by the fact it doesn't depend on hand crafted heuristics. It differs as such our work can be trained once on more than a room type, it wouldn't require a model to be trained for each room type.

Another approach was introduced by [Wan+19], where they divide their approach into two stages, a planning one and an instantiating one, they represent the scene as a relation graph in a similar way. Objects are the nodes and the relationships between them are the edges.

In the planning stage, it uses a deep graph Convolutional Generative Model to synthesize relation graphs. And in the instantiating stage, it uses image-based CNN modules to place objects into the scene in a manner consistent with the graph. The difference between this approach and our, is that our work deals with another representations of the scenes. While this work have good results, they are just images at the end. The output of our work in a completely annotated 3D scene.

Plan2Scene [Vid+21] came out recently, it converts 2D floor plans to 3D scene, the main focus of this work was the texture synthesis and not interior design and placing objects in the scene as this work. While in our work we have introduced a room styling module, the main focus was to get plausible scenes from rooms layouts.

3 Our Approach

Transformer networks and their components form the core of our work. In this section, a discussion of applying the task of 3D indoor scene synthesis that is based on transformer architecture is done. There are five networks that are responsible for five different tasks, namely the object's category, orientation, location, dimension, and texture style. These networks and their variations are discussed in the following.

3.1 Transformer for scene synthesis

In this section, our first approach on how transformers can be used for the task of scene synthesis is discussed. Transformers are used to handle sequential data [Vas+17], so reformulating the scene synthesis problem to a sequence generation problem is what should enable the transformer to process the 3D scene data.

Inspired by [WYN20], the scene is represented by the sequence of the objects it has. This sequence is ordered from the most frequent object to the less frequent one. For the transformer to answer the questions of "what" and "where" it needs to be trained on a sequence that has the answers to those questions. Making our problem a **supervised learning problem**. So for each object in the scene, sorted by frequency, "what" kind of object it is, namely the category and "where" it is namely the 6D pose. These categories and poses are then used as our training data.

3.1.1 Tokenization

Transformers work with discrete values, and so the input values for a category or 6D pose have to be quantized. For example, the object location in a scene could take infinitely many values. Transformers are not capable of working with such types of unconstrained values. Therefore the object's 6D pose and category are quantized into discrete numbers for the transformer to work with them.

Each category is given a unique identifier which here is called a token.

For the location values, they are scaled by the maximum room size and quantized into 80 discrete categories. For the object orientation values are converted to degrees angles, meaning 360 tokens for the orientation.

In order for our model to learn those token, they need to be presented in convenient way for the transformer. Representing tokens in an one-hot encoding is common in the neural language processing field. However, when the vocabulary of words becomes large, a more efficient representation is needed. Which is the reason the embedding layer is added.

3.1.2 Embedding

The task of the embedding layer is to learn to map our tokens to latent vectors, possibly finding a unique vector for each word in a much-lower dimensional space. Making this approach much more efficient than the one-hot encoding of the tokens.

A example of the usage of two embedding layers is shown in figure 3.1. The category and orientation sequences of a room are embedded. For the category sequence, 41 is start token, 42 is the padding token, 43 is the stop token. The values between the start token and the first padding token are the objects categories tokens. The number of padding tokens is calculated by the following formula:

$$pad = lmax + lobjects - 2$$

So the count of padding tokens is equal to, maximum sequence length - length of objects in the room - 2 (for start and stop tokens).

For the orientation sequence, 361 is start token, 362 is the padding token, 363 is the stop token.

The result vector is then summed with the positional encoding and the contextual information which are provided to the transformer.

3.1.3 Context

As explained in chapter transformer , the main two components of a transformer are an encoder and a decoder, the function of each encoder layer is to generate encodings that contain information about which parts of the inputs are relevant to each other, while each decoder

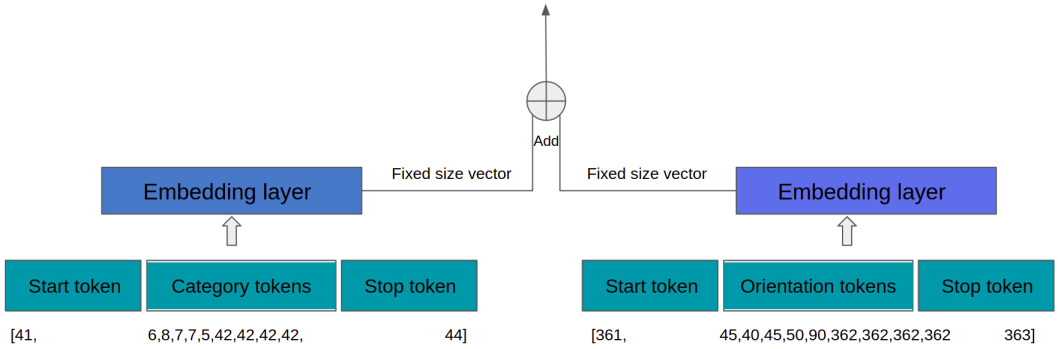


Figure 3.1: The task of the embedding layer is to learn to map our tokens to vectors, possibly finding a unique vector for each word in a much-lower dimensional space. For the category sequence, 41 is start token, 42 is the padding token, 43 is the stop token. The values between the start token and the first padding token are the objects categories tokens. For the orientation sequence, 361 is start token, 362 is the padding token, 363 is the stop token. The values between the start token and the first padding token are the objects orientation tokens.

layer does the opposite, taking all the encodings and using their incorporated contextual information to generate an output sequence.

In order for the transformer to learn poses and classes of objects, it needs some information about the shape of the interior space, like where the walls are. Hence providing the transformer with awareness of the room shape, is a must for the model to learn.

The Room layout is presented as a one channel 2D array, and to present it in a way consistent with transformer some inspiration from the work of [Nas+20] conditioning the model on an image of the room layout is taken. As shown in figure 3.2, the room layout, as an image is fed to a small Resnet [He+16] to extend some features. The features are then flattened, and pixel indexes are embedded and added to the features.

This work further introduces a single model, that is able to learn multiple classes of rooms. This is the first work that does this given our research on the topic. The room layout image is colored by a specific color for each room type it belongs to as shown in figure 3.3.

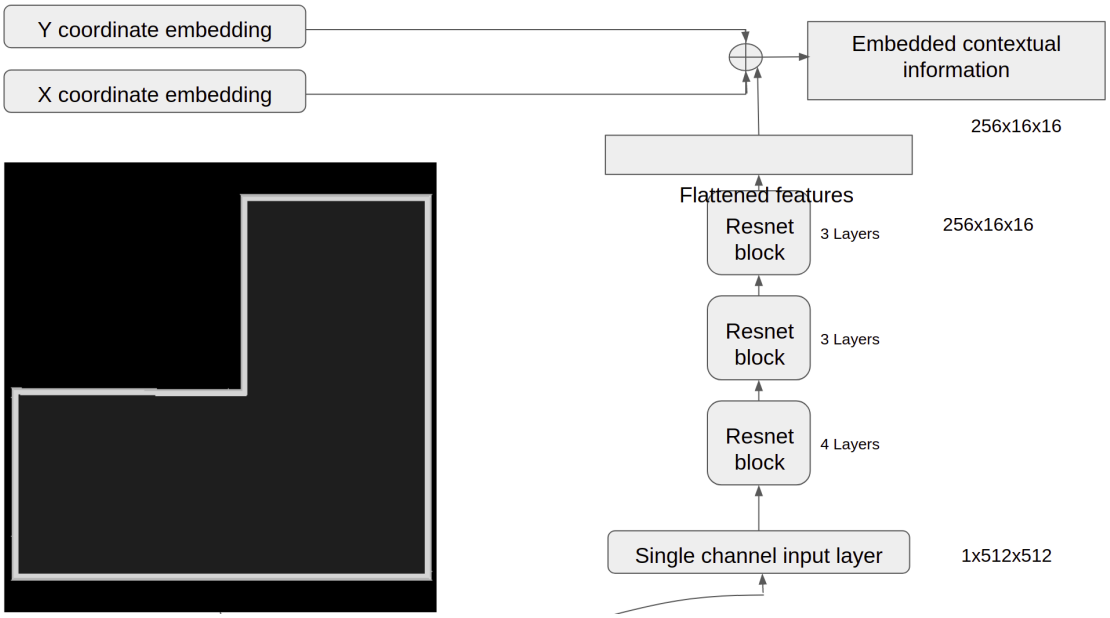


Figure 3.2: How the network deal with the room shape as contextual information. One channel image of the room layout is passed through three Resnet [He+16] blocks. The resulting features are flattened and added to the pixel coordinates embedding, inspired by [Nas+20]

3.2 The EnvironmentFormer

Figure 3.4 shows the full transformer model which is used for one of the tasks, namely category prediction task, with small changes in the model from one task to another.

Positional encoding is directly the position of the object in the sequence. This position is based on the object frequency in this across the whole dataset for same type of room. This was chosen so the model would predict frequent objects more than less frequent ones.

The encoder and decoder blocks were repeated eight times as the original [Vas+17]. The input sequences of tokens are converted to a fixed size vectors using embedding layers and summed together. The results are then added with the positional encoding of each token in the sequence.

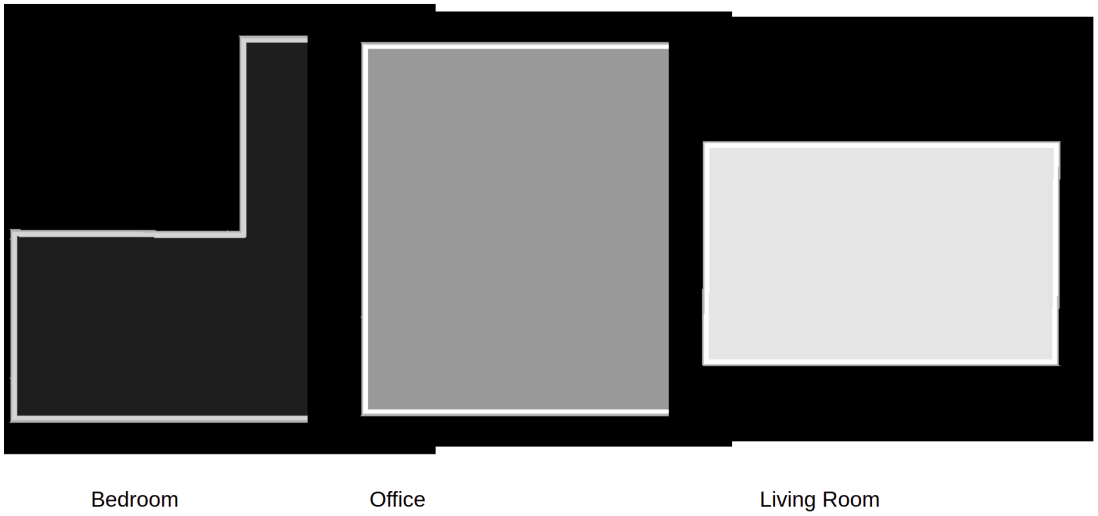


Figure 3.3: This figures shows how our work deals with multiple room types, and how to encode this information. One channel image of the room layout is passed through three Resnet [He+16] blocks. The resulting features are flattened and added to the pixel coordinates embedding, inspired by [Nas+20]. Giving a specific color for the layout of each room type is what enables the model to learn multiple classes of room.

3.3 Category EnvironmentFormer

In this section we explain the Category EnvironmentFormer, the EnvironmentFormer task is explained by discussing the inputs and outputs of the EnvironmentFormer.

Inputs starts with the category sequence, since it is the target. This work investigates which sequences could be useful for our EnvironmentFormer to learn categories with high accuracy.

For the category sequence, choosing only the forty most frequent categories, because the frequency of finding an occurrence from one of the next categories was very low, less than 1% [Son+17]. So the model would have a start token of 41, stop token of 42, and a padding token of 43.

At the beginning of training or testing, the first category token is the category start token. Each time the model decide to add an object to the scene, this object's tokens are added to

the corresponding sequences. Dealing with the problem in an auto regressive way. The same goes for the orientation sequence as well, except the model doesn't learn it.

The location and dimension has three times the information per object compared to category or orientation (there is only one orientation value which is needed, and which is used in the sequence, explained 3.4). So for location and dimension input sequences, the model needs three embedding layers for each.

The categories are obtained by mapping the original [Son+17] to the [NF12] categories, because they provide a higher level of categorization.

Outputs is a categorical tensor of the top 20% object category IDs the transformer suggests to be added next.

3.4 Orientation transformer

Inputs: the orientation sequence, since it is the target. As expected, after doing some experiments with adding other sequences to the inputs, the model with the addition of location, category and dimension sequences performed better than all other variants that was tested, as shown in the results section 5.0.2. The orientation is chosen to have single token because in our problem, there is only one orientation angle that matters. Placing an object in a room, one would only need one orientation angle, because the object would need to be placed on the floor, which constrains the other two degrees of freedom. Hence making the orientation problem easier for our model. During the prepossessing step, the object orientation is converted to degrees. The tokens are chosen to have 360 values, as to match the degree angles directly.

Following the same procedure as in 3.3, the first token is the orientation start token. Each time the model decide to add an object to the scene, this object's tokens are added to the corresponding sequences. Dealing with the problem in an auto regressive way. The same goes for the category sequence as well, except this model doesn't learn it.

The location and dimension has three times the information per object compared to category. So for location and dimension input sequences, the model needs three embedding layers for each, same as the category model.

Outputs is a categorical tensor of the top 20% object orientation in degrees.

3.5 Location transformer

Inputs: the location sequence, because it is the target. Using a model that knows all the possible information about the object made the most sense. Learning the location is the most difficult task out of the four of, category, orientation and dimension. It has three times the data compared to the first two, and more possible combination of values than the third.

To have a consistent location values across all room, so the model can learn. A maximum room size is set and the location if the object is scaled by that size. Then the result of that is quantized to the maximum number of tokens, eighty tokens were used for location values.

Following the same procedure as in 3.3, the first token is the location start token. Each time the model decide to add an object to the scene, this object's tokens are added to the corresponding sequences. Dealing with the problem in an auto regressive way. The same goes for the category and the other two sequences as well, except this model doesn't learn them.

The location is three times the information of the category and orientation, so it has three times the sequence length. To be able to add the sequences they must the same size. The solution for this problem, inspired by [WYN20], was repeating both the category and orientation sequence three times. Hence ending up with same size sequences, and the transformer was still able to learn.

Outputs is a categorical tensor of the top 30% object locations.

3.6 Dimension transformer

In principle, the dimension transformer is the same as the location transformer 3.5.

Inputs: They key difference from the location transformer 3.5, is that the dimension model learns the dimension instead of location. The loss is calculated based on the dimension sequence instead of the location one.

The dimension is as well three times the information of the category and orientation, so it has three times the sequence length. To be able to add the sequences they must the same size. Using the same approach as used in the location transformer 3.5, category and orientation sequences were repeated three times. Hence ending up with same size sequences, and the transformer was still able to learn.

Outputs is a categorical tensor of the top 30% object dimensions.

3.7 Room Generation algorithm

In this section, the process of combining everything that was discussed so far in a single algorithm. This algorithm is what is used to generate rooms.

The process of producing a room is divided into the following steps:

1. The input is a room type and a room shape.
2. An empty room is created using the input room shape.
3. The room type and shape are combined and represented as a single vector.
4. This vector is an input to all the different EnvironmentFormers.
5. The category EnvironmentFormer is the first one to be invoked. If it is the first object, the input is the start token of each sequence this EnvironmentFormer takes. Else the inputs are the sequences of the current objects which are already added. The position for the positional encoding layer always points to the start token of the category sequence.
6. Using the category which was produced by the first EnvironmentFormer, the Orientation EnvironmentFormer is invoked. The category token, which is produced from the previous step is used. The orientation, location and dimension tokens are the start token of each sequence.
    a) The location EnvironmentFormer is invoked three times. Once per coordinate, x, y and z. The EnvironmentFormer takes the position in the sequence of the object to infer its location. To get x, y and z for each run, the position of the object is incremented. For example the position of the object in the sequence is (object position in the sequence + 0) for x coordinate, (object position in the sequence + 1) for y and (object position in the sequence + 2) for z. The location sequence is three times the length of the category or orientation sequence.
    b) The output of the location EnvironmentFormer is filtered and only the top 20% locations are considered.
        i. Randomly sample one of the top 20% locations.
        ii. The dimension EnvironmentFormer is invoked three times. The same tokens that are the input to the location EnvironmentFormer are used for the dimen-

sion EnvironmentFormer, except for the location tokens. The location tokens that are used in this step are now the result from the previous step.

iii. Use the dimension model to choose a model from the 3D models set.

iv. If there is no collision in the room, the model is placed. Else another location is sampled and the dimension model is invoked again with the new location.

v. If the re positioning trails exceed maximum re positioning trails, the algorithm stops.

3.8 Texture styling model

A neural style transfer network is integrated into this work, the network is based on [GEB15a] finding, that the representations of content and style in the Convolutional Neural Network are separable. When Convolutional Neural Networks are trained on object recognition, they develop a representation of the image that makes object information increasingly explicit along the processing hierarchy [GEB15b]. Which means, along the depth of the CNN, the input image is transformed into a representation that care more about the actual and high level content, instead of pixel level values.

And for the representation of the style of an input image, the model uses a feature space originally designed to capture texture information [GEB15b]. This feature space is built on top of the filter responses in each layer of the network. It consists of the correlations between the different filter responses over the spatial extent of the feature maps. By including the feature correlations of multiple layers, the result is a stationary representation of the input image, which captures its texture information but not the global arrangement.

The key for this is defining two distances, one for the content (**Dc**) and one for the style (**Ds**). After loading our target texture image, it is transformed to minimize both its content distance with the itself and its style distance with the style image.

The figure 3.6 from [GEB15a] visualizes the network. It explains with an example how a style and a content image are dealt with by the CNN. It also visualize what the results of the intermediate stages of processing.

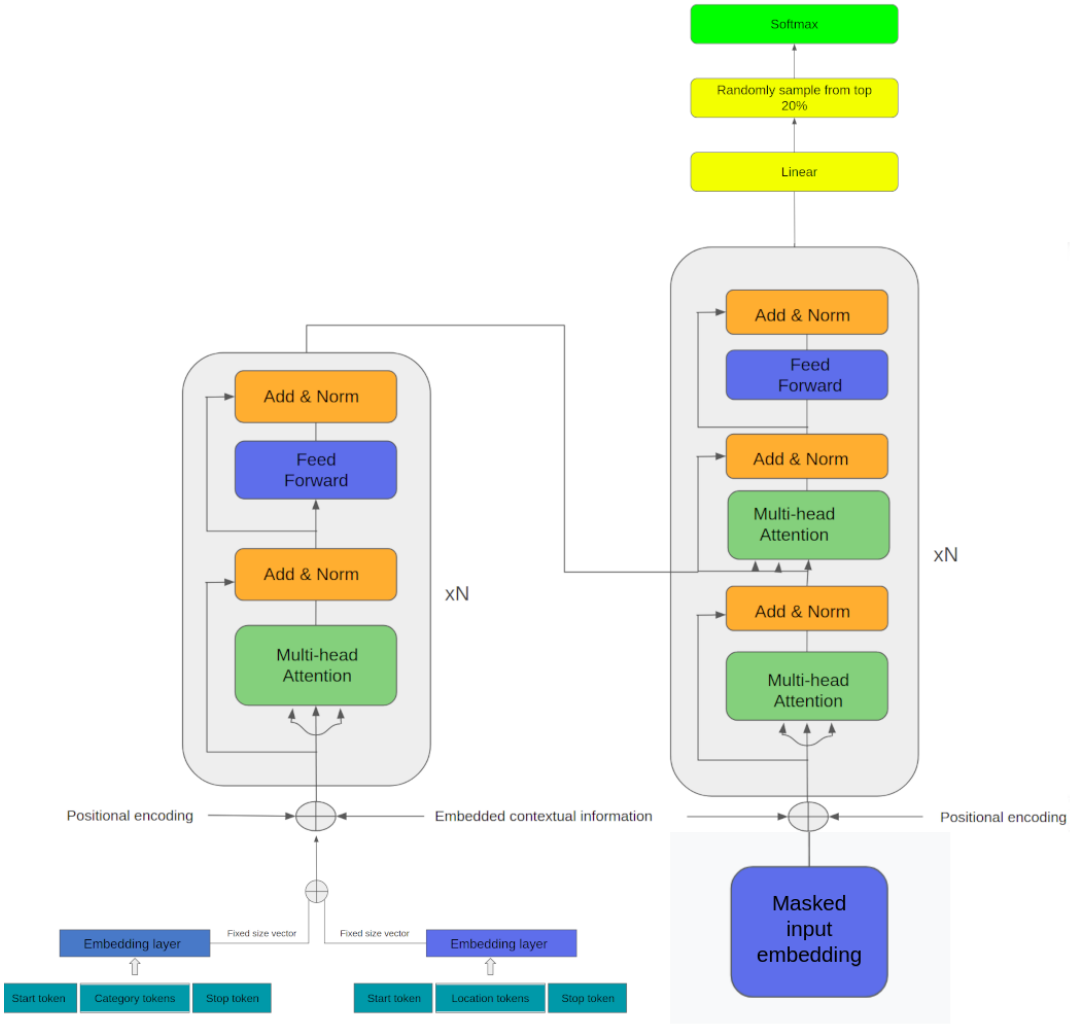


Figure 3.4: Our EnvironmentFormer model. N is, the number of how often this block is repeated. The input sequences of tokens are embedded and added together. They are then added with the positional encoding of each token in the sequence. The objects in the scene are sorted based on frequency across all data set for this specific room type. Depending on the type of the model, the contextual information are then added with the latter before going to the encoder. The same embedded vector goes through the decoder with the exception that the future tokens are masked so it can learn the sequence.

Figure 3.5: Neural Style Transfer, allows you to take an image and reproduce it with a new artistic style. The algorithm takes three images, an input image, a content-image, and a style-image, and changes the input to resemble the content of the content-image and the artistic style of the style-image [Pas+19].

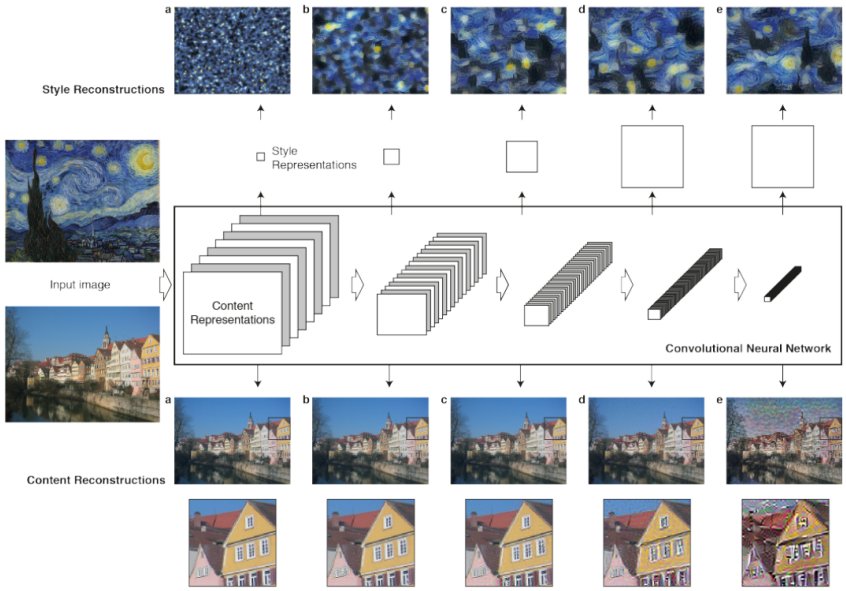


Figure 3.6: Convolutional Neural Network (CNN). Style Reconstructions. On top of the original CNN representations we built a new feature space that captures the style of an input image. The style representation computes correlations between the different features in different layers of the CNN. We reconstruct the style of the input image from style representations built on different subsets of CNN layers (‘conv1 1’ (a), ‘conv1 1’ and ‘conv2 1’ (b), ‘conv1 1’, ‘conv2 1’ and ‘conv3 1’ (c), ‘conv1 1’, ‘conv2 1’, ‘conv3 1’ and ‘conv4 1’ (d), ‘conv1 1’, ‘conv2 1’, ‘conv3 1’, ‘conv4 1’ and ‘conv5 1’ (e)). This creates images that match the style of a given image on an increasing scale while discarding information of the global arrangement of the scene [GEB15a].

4 Experimental Setup

4.1 Environment

For our work, the GPU cluster provided by the Institute for Robotics and Mechatronics of the DLR (German Aerospace Center), is used. The cluster consists of Titan V 12GB, Titan RTX 24GB, 2080 RTX Ti 11GB and Quadro GV100 32GB GPU's. Most of our training and testing is carried out on a single Titan V 12GB GPU.

Our code for the work is implemented using Python 3.8 and Torch 1.4 [Pas+19] is used for all Deep Learning-related GPU operations. PyTorch lightning [Fal19] is a lightweight PyTorch wrapper for high-performance Machine Learning research. In our code PyTorch-Lightning is used to wrap training methods.

4.2 Datasets

Datasets are important in any learning-based method. With the rise in the application of Deep Learning in almost every field, there has been quite a steep increase in the number of public datasets. But, when it comes to datasets of annotated 3D environments, which have accurate 3D data of the objects in the environments including normal information, location, orientation, and size. The amount of those datasets has not been increasing at the same pace. This happens as those types of datasets are difficult and costly to build as explained in the related work.

4.2.1 SUNCG

SUNCG dataset is a large-scale dataset of synthetic indoor 3D scenes with dense object centered annotations [Son+17]

The pre-processing of the dataset is done using a modified version of the data creation scripts from [RWL18], choosing just the rooms that contain a floor node. This leads to the

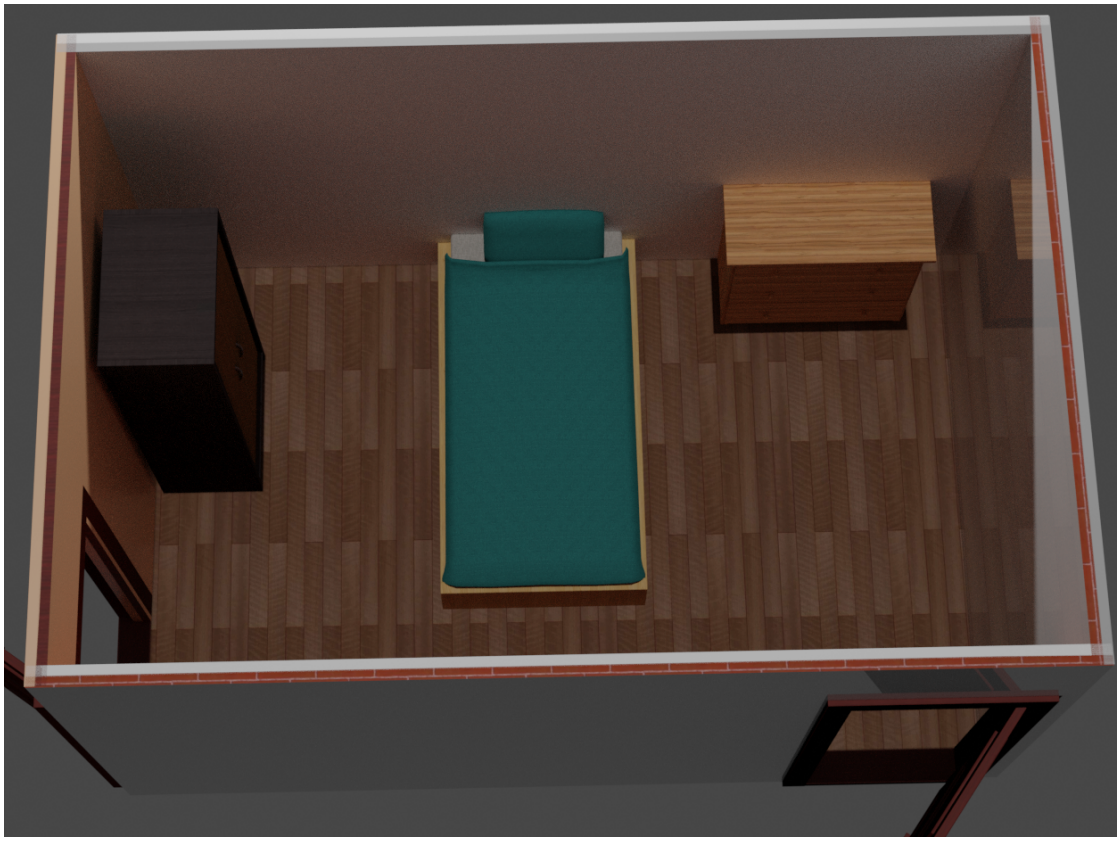


Figure 4.1: A bedroom from SUNCG dataset [Son+17] rendered with [Com18]. This bedroom was‘ part of our training data.

creation of 2434 valid bedrooms, 930 kitchen and so on as shown in figure 4.4. Given that the most valid room types are bedrooms, it became the chosen room type for single type room models experiments.

Figure 4.2: Sample from the offices in the SUNCG dataset [Son+17] rendered with [Com18].

Room type	Rooms
Bedroom	2434
Kitchen	1863
Toilet	1601
Hall	473
Office	451
Living room	313
Balcony	305

Figure 4.4: Rooms after our preprocessing and filtering in SUNCG 4.2.1 dataset.

4.3 Data preprocessing

The preprocessing steps of SUNCG data 4.2.1 is explained in this section.

Utilizing the scripts provided by [RWL18]. Those preprocessing scripts were used to filter the rooms. They filtered by room type, which was necessary for our work. Filtered out

Figure 4.3: Sample from the offices in the SUNCG dataset [Son+17] rendered with [Com18].

rooms that are missing key components, like walls and a floor. Rendered room layouts, which is the top down view of each room, a bedroom example is visualized in figure 4.5. This layout is used to extract contextual information as explained in section 3.1.3.

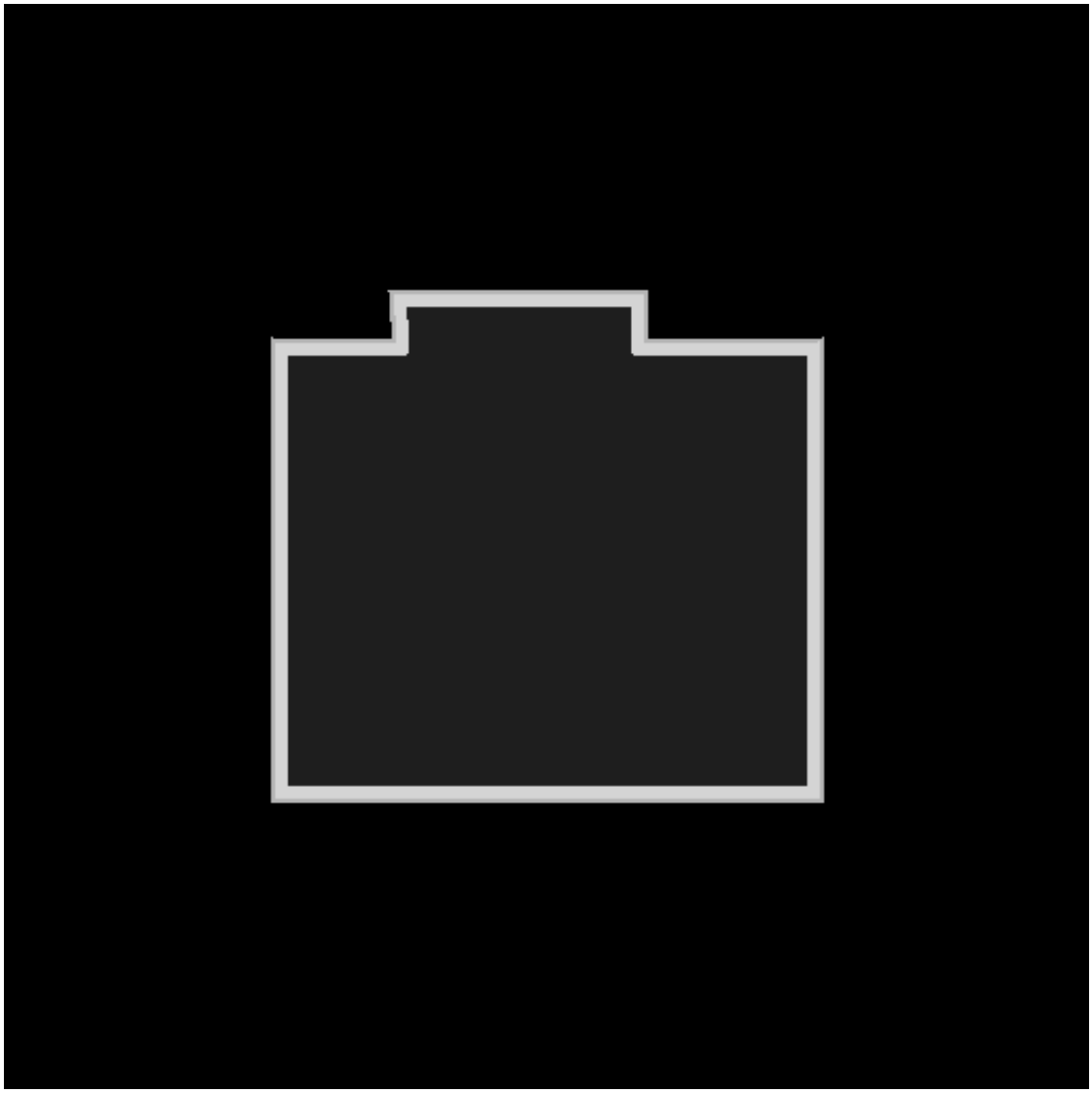


Figure 4.5: A bedroom layout, which is used to provide our transformer with the room shape information, as explained in section 3.2.

The location and orientation information could be directly processed from the 4.2.1 json files. The category of each object model could be mapped by using resources provided by 4.2.1.

To write the dimension information of each object we used Blender API. Loading each object into blender then directly writing the object dimension to a dictionary.

Objects in each room are sorted by their frequency in the whole dataset. The positional

encoding part of the network needs a position of each object as explained in section 3.2. So a script was written to save a dictionary that has all room types and the frequency of objects categories in these rooms. Figures 4.6 and 4.7 shows our findings about which objects are more common in certain room types, and which objects are less common.

Frequency	Category
1	Bed
2	Wardrobe Cabinet
3	Stand
4	Chair
5	Rug
6	Desk
7	Dresser
8	Dressing Table
8	Curtain
9	Ottoman
10	TV Stand
11	Indoor Lamp
12	Table
13	Shelving
14	Plant
15	Sofa
16	Music
17	Mirror
18	Person
19	Shoes
20	Hanger

Figure 4.6: Twenty most frequent object categories in bedrooms in 4.2.1 dataset.

Frequency	Category
1	Sofa
2	Window
3	Table
4	Plant
5	Chair
6	TV Set
7	Rug
8	Indoor Lamp
8	Curtain
9	Music
10	Shelving
11	Ottoman
12	Fire Place
13	Wardrobe Cabinet
14	Toy
15	Vase
16	Sand
17	Picture Frame
18	Dresser
19	Column
20	Partition

Figure 4.7: Twenty most frequent object categories in Living rooms in 4.2.1 dataset.

4.4 Models

For Transformer and Convolutional Neural Network models, the model architecture and hyper parameters have a huge impact on the performance. Which is the topic of many ongoing and emerging research [Kha+19]. In our work, the focus was not to build a new transformer architecture or a new style transfer neural network, but rather the adoption of these models to solve our problem which is explained in section section 1.1.

In this work, three models are used. Two of them for scene generation, one for coloring and styling textures. The main model is the transformer network which is explained in section **??** in details. The usage of the transformer in our work is further explained in section 3.2.

Both the second and third models are Convolutional Neural Network. Our second model was a Renet **HeZRS15**. The model was used to extract features that were added later to our transformer as explained in section 3.1.3. The third and final model used in our work was a Neural Style Transfer Network [GEB15a]. The Network is used for coloring textures and explained in details in section3.8.

4.4.1 Transformer

The Transformer was used for four main tasks. The tasks are Category, location, orientation and dimension prediction. Our transformer was discussed in details in section 3.2. In this section, our transformer process and hyper parameters are explained.

Trying not to converge away from [Vas+17] original architecture, the hyper parameter are chosen to be the same, when possible. The number of heads was chosen according to that. For the embedding dimension, 256 was chosen for the category and orientation transformer instead of 512. The decrease in embedding dimension was because the extra 256 length provided no additional performance for those two models. Those two tasks were simpler and the information could be represented in a 256 vector. The transformers hyper parameters are shown in tables 4.8 and 4.9.

The maximum sequence length is determined by the maximum number of object that are processed per room. The sequence length is then the number of token of all sequences in the transformer. For category and orientation transformers this value is (start token + maximum number of object that are processed per room + stop token), having twenty four as a result. While for the location and dimension transformers, the sequences are three times longer, for example it would be, (**start token** + ((x-value + y-value + z-value) x maximum number of object that are processed per room) + **stop token**).

To solve the problem of not having the same sequence lengths for all information, the category and orientation sequences are copied twice to match size, this part was explained in section. 3.3. The result sequence would look like (**start token** + ((category token of object) x three) x maximum number of object that are processed per room) + **stop token**).

Loss

The loss function used by our transformer is the cross entropy loss. Which is the common loss function used in transformer models [Lin+21].

$$CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{\mathrm{N}_c} y_i \log(\hat{y}_i) \tag{4.1}$$

where $\boldsymbol{y} \in \mathbb{R}^1$ is a vector of length embedding size and N_o is the number of objects in the sequence.

Hyper parameters	
Embedding dimension	256
Maximum sequence length	24
Heads for Multi-head attention	8

Figure 4.8: Transformer hyper parameters for category and orientation models.

Hyper parameters	
Embedding dimensions	512
Maximum sequence length	64
Heads for Multi-head attention	8

Figure 4.9: Transformer hyper parameters for location and dimension models.

4.5 Training

In this section the training parameters for our EnvironmentFormers are shown.

Table 4.1 shows the hyper parameters for our EnvironmentFormers, in case of training a single room type model, of type Bedroom.

Table 4.1: This table shows the hyper parameters for our EnvironmentFormers, iin case of training a single room type model, of type Bedroom.

Hyper parameters	
Learning rate	0.0003
Embedding dimensions	256
Objects per room	20
Drop out	0.3
Epochs	600
Training data length	2214
Validation data length	246

4.6 Implementation

Each room is stored as a single pickle dictionary. For constructing the scene and collision checking, version 2.37.13 of Trimesh [Daw] was used.

5 Experiments

In this chapter the experiments are shown and discussed. Those experiments helped by giving hints to solve the design problems that are faced. It begins by the Category EnvironmentFormer, then the Orientation EnvironmentFormer experiments. After that, the Location and Dimension EnvironmentFormer are related experiments are presented.

5.0.1 Category EnvironmentFormer

The Category EnvironmentFormer task is to predict the possible categories that could be added to the scene next. The input is a sequence of tokens of all the already present objects in the scene.

Basic Category EnvironmentFormer

Our EnvironmentFormer is trained on just the category sequence as an input. Figure 6.7 shows the result of training this category EnvironmentFormer.

Orientation and location aware category EnvironmentFormer

In this variant of category EnvironmentFormer, the orientation and the location sequences are provided as well. This variant gave lower loss, which means the location information was useful for the category EnvironmentFormer.

Orientation, location and dimension aware category EnvironmentFormer

In this variant of the category EnvironmentFormer, the orientation, the location and the dimension sequences are provided as well. This variant gave similar and close results to the Orientation and location aware variant. Meaning the dimension sequence doesn't hold much useful information for the category EnvironmentFormer.

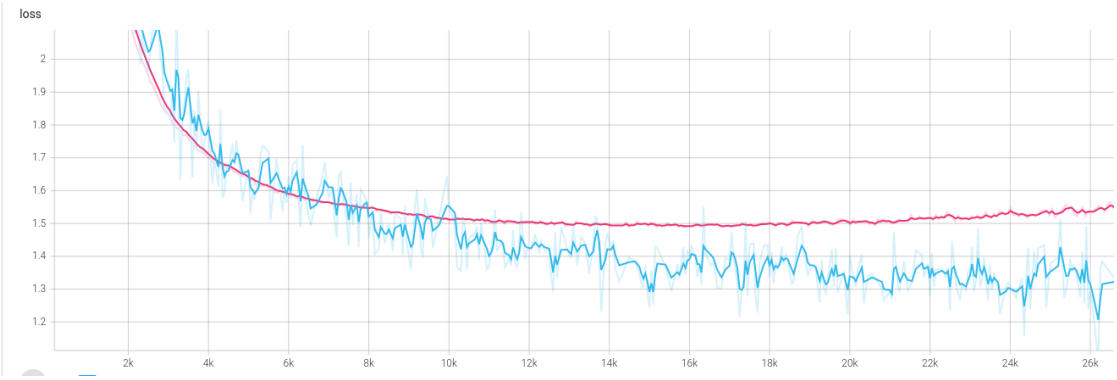


Figure 5.1: This graph shows the training loss in blue, and the validation loss in pink, per step for our category transformer. The lowest validation loss was 1.5. this loss was achieved after 13,000 steps. After 13,000 steps our transformer started to overfit.

5.0.2 Orientation EnvironmentFormer

This EnvironmentFormer task is to predict the possible orientations of an object given its category. The EnvironmentFormer also takes into account all the objects already present in the scene (auto regressive).

Location and dimension aware orientation EnvironmentFormer

In this variant of orientation EnvironmentFormer, the location, the category and the dimension sequences are provided as well.

The orientation EnvironmentFormer is trained once with 90 tokens, and once with 360 tokens. The EnvironmentFormer which is trained with 90 tokens performed better, and converged faster. Concluding that, 90 token is enough to represent degree angles and learn them. However what is gained in performance in the EnvironmentFormer, is lost in precision. Each token has a size of four degrees. A simple multiplication is used to interpolate the results back into degrees.

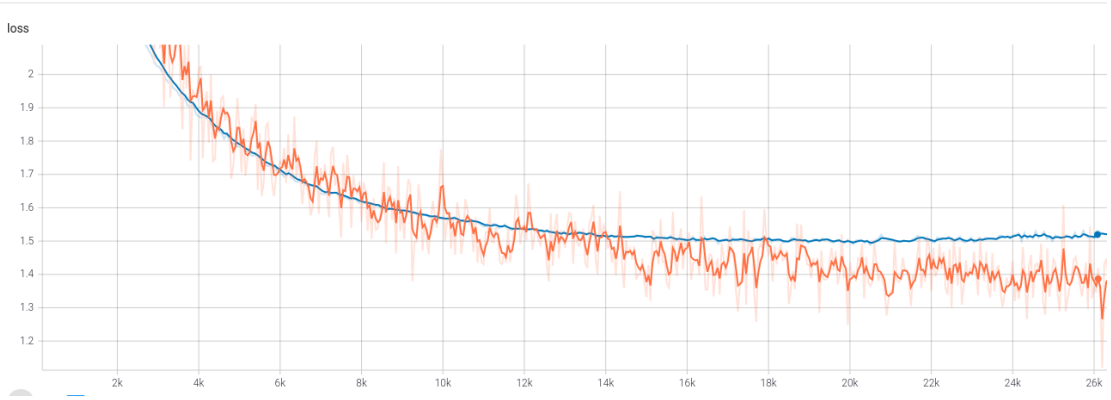


Figure 5.2: This graph shows the training loss in orange, and the validation loss in blue, per step for our category transformer. The lowest validation loss was 1.48. This loss was achieved after 22,000 steps. After 22,000 steps our transformer started to overfit.

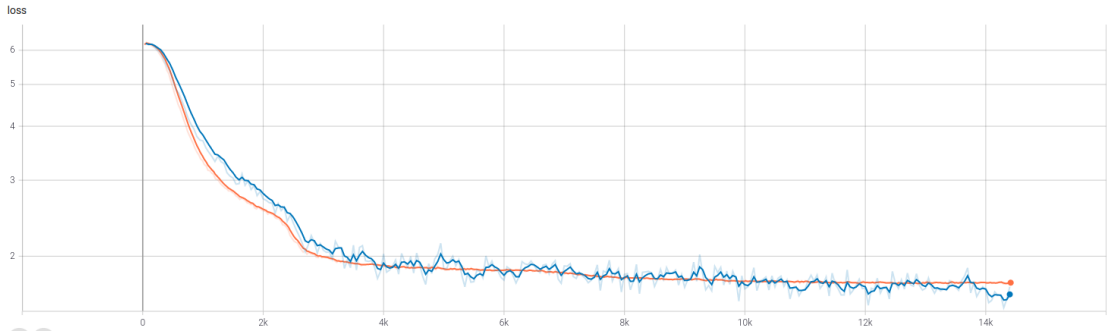


Figure 5.4: Orientation EnvironmentFormer trained with category, location and dimension sequences. The validation loss is in orange, the training loss is in blue. There are 360 used. The EnvironmentFormer started to converge after 14 thousand steps with a validation loss of 1.735.

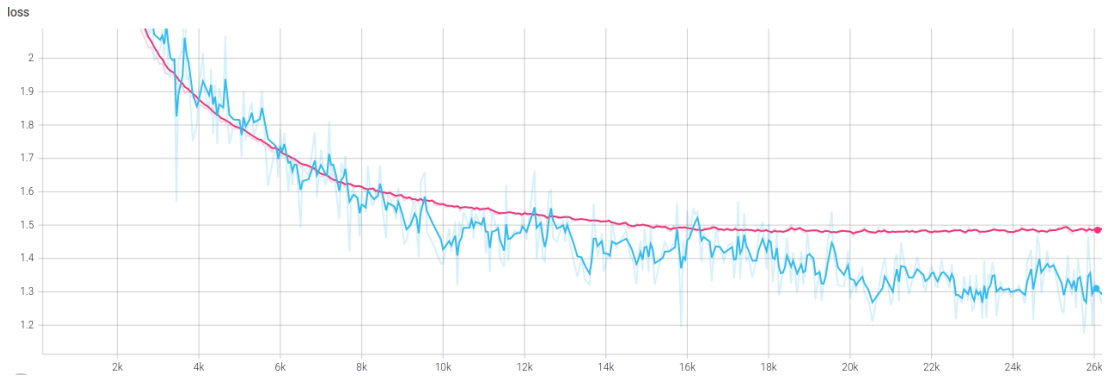


Figure 5.3: This graph shows the training loss in orange, and the validation loss in blue, per step for our category transformer. The lowest validation loss was 1.47. This loss was achieved after 20,000 steps. After 20,000 steps our transformer started to overfit.

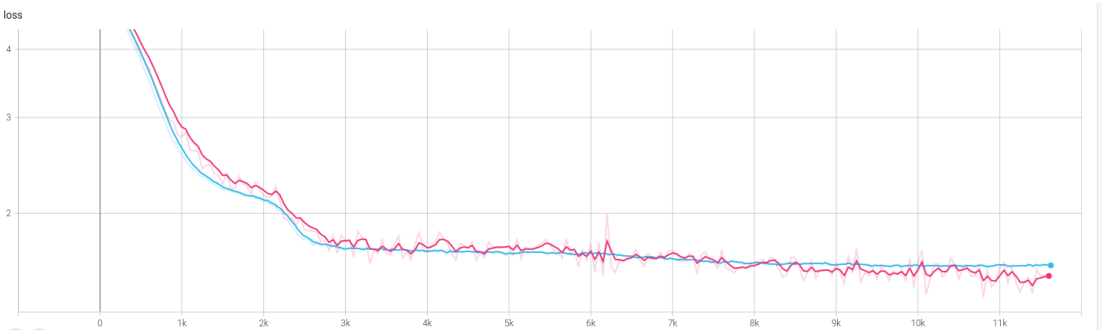


Figure 5.5: Orientation EnvironmentFormer trained with category, location and dimension sequences. The validation loss is in blue, the training loss is in red. There are 90 tokens used. The EnvironmentFormer started to converge after 11 thousand steps with a validation loss of 1.6.

5.0.3 Location EnvironmentFormer

This EnvironmentFormer task is to predict the possible locations of an object given its category and orientation. The EnvironmentFormer also takes into account all the objects already present in the scene (auto regressive).

Dimension aware location EnvironmentFormer

In this EnvironmentFormer variant, the dimension sequence is also a part of this EnvironmentFormer input. This EnvironmentFormer achieved lower validation loss. Which means the dimension information of the objects in the scene provide a hint for our location EnvironmentFormer. This hint is used to predict where to place an object more accurately.

Two loss graphs for location EnvironmentFormer with dimension sequence as input, and another with no dimension sequence as input, are shown below respectively.

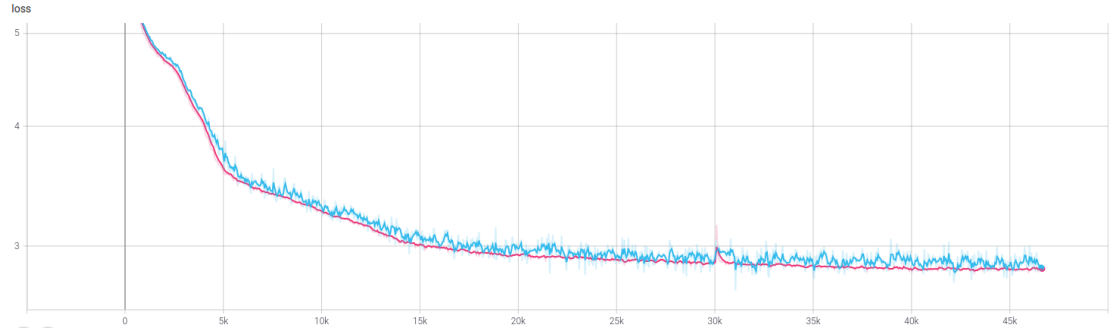


Figure 5.6: Location EnvironmentFormer trained with category, orientation and dimension sequences. The validation loss is in red, the training loss is in blue. The tokens used were 80. The EnvironmentFormer converged, and after 20 thousand steps it had a validation loss of 2.83.

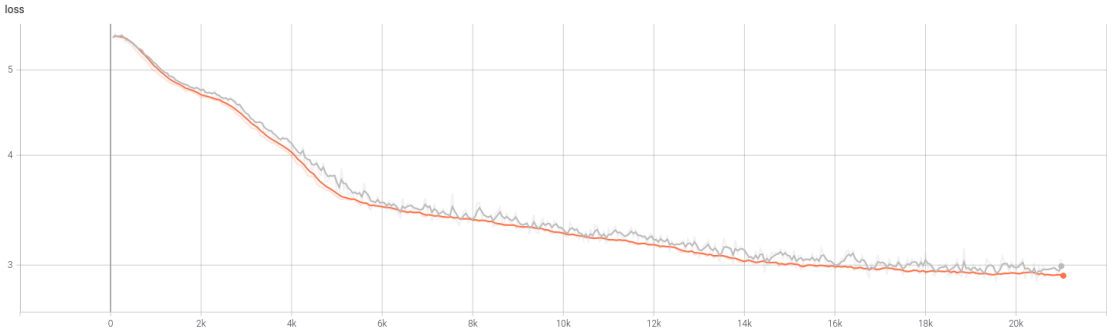


Figure 5.7: Location EnvironmentFormer trained with category and orientation sequences. The validation loss is in orange, the training loss is in grey. The tokens used were 80. The EnvironmentFormer converged, and after 20 thousand steps it had a validation loss of 2.9.

5.0.4 Dimension EnvironmentFormers

This EnvironmentFormer task is to predict the possible dimensions of an object given its category, orientation and location. The EnvironmentFormer also takes into account all the objects already present in the scene (auto regressive). The complete category, orientation, and location sequences act as an input to this EnvironmentFormer.

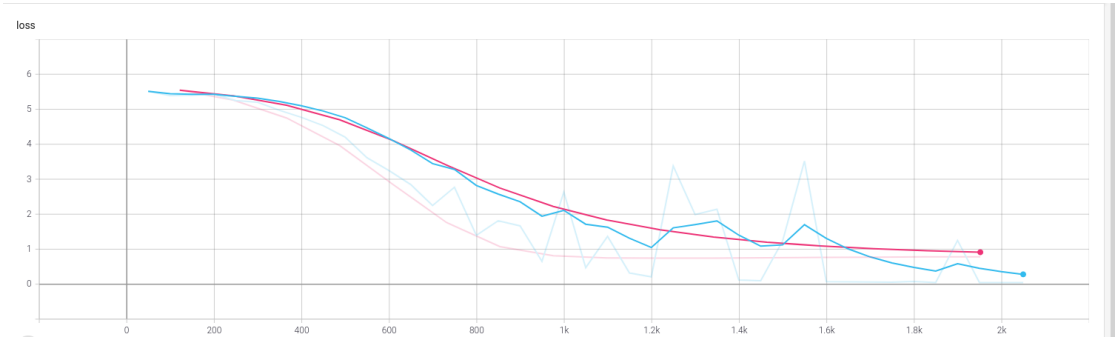


Figure 5.8: Dimension EnvironmentFormer trained with category, orientation and location sequences. The validation loss is in red, the training loss is in blue. There are 90 tokens used, same as location model. The EnvironmentFormer converged, and after 20 thousand steps it had a validation loss of 0.4.

6 Evaluation

In this chapter, the results of our work are presented. The results are also compared to other related works. The results of training of the four EnvironmentFormers combined are discussed. Then the results of applying a neural style transfer network to the textures is shown.

The results for a single room type EnvironmentFormers are presented. Followed by the results for multiple room types EnvironmentFormers.

6.1 Single room type results

The results of a single room type are first presented. These results are for the bedroom type. The bedrooms were chosen because they had an average of more than 14 objects per room, while also being the most rooms in the SUNCG dataset [Son+17].

The input to our algorithm is a room layout. Doors and windows are considered part of the room layout, so they are treated as inputs as well.

Figure 6.1 shows an example of a room layout, which is what this algorithm needs to generate the result shown in the next figure.

The bedroom in figure 6.2 was generated by running the full algorithm as explained in section 4.7. The four EnvironmentFormers seem to have learned the symmetry of objects around a bed in a bedroom. To position the objects in this room, our algorithm sampled the location from the location EnvironmentFormers once, for the bed, and once for the stand, and three times for the wardrobe. Then after reaching maximum re-posistioning trials, there was no space to add another object that would not violate the collision threshold. Which means our room is complete.

The room contains a bed, a wardrobe cabinet and a bed stand. Those objects are very common to find in a bedroom.

A bedroom generated by running the full algorithm as explained in section 4.7. The room has a bed, a stand and a wardrobe cabinet.

Figure 6.1: A room layout, an example for the input of the algorithm.

The closest bedroom in the dataset to the room generated in 6.3 is shown in figure 6.4. This room has the same layout as the generated one, and is part of the algorithm training data.

While this room 6.5 had 5 doors, it can still be recognized as a bedroom. Here there is a sink in front of a chair in a bedroom, this shows some of the limitations in the category EnvironmentFormer of this algorithm.

The symmetry around the bed holds. Some limitation of the location EnvironmentFormer is shown in the position of the Chair. Not all the objects are in contact with the floor, some are floating like the rug and the wardrobe cabinet on the left of the bed.

Even though this result show that the EnvironmentFormers are powerful and work with non uniform room shapes. It shows some of the limitations of the location EnvironmentFormer appears in this result. A bed shouldn't be in front of the door like this. The wardrobe and the bed are floating as well.

6.2 Room type encoded

The results of multiple room types sharing weights in a single EnvironmentFormer are first. These results are for the bedroom, office and living room types.

Three types of rooms were chosen randomly. Mixing between room types that has small

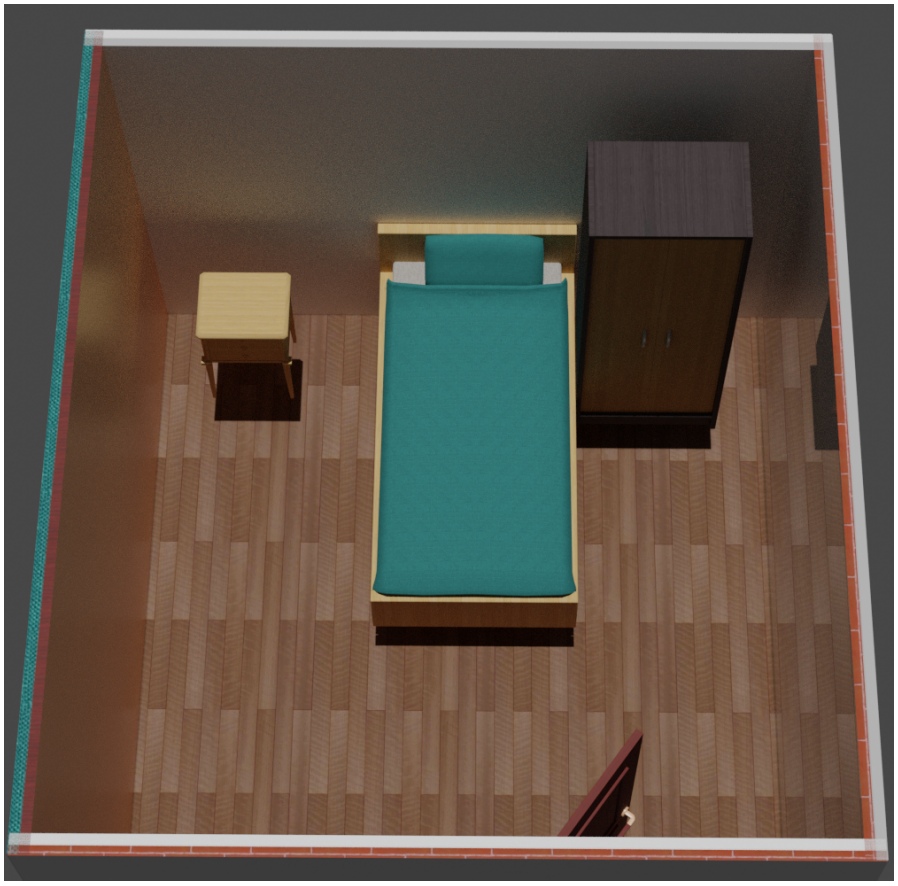


Figure 6.2: A bedroom generated by running the our algorithm as explained. The room contains a bed, a wardrobe cabinet and a bed stand.

average of objects per room, and the ones with very high average of objects per room resulted in a bad performance for our EnvironmentFormer. For this reason bedrooms, offices and living rooms are used to test our novel approach.

The input to our algorithm is a room layout, and a room type. Doors and windows are considered part of the room layout, so they are treated as inputs as well.

Figure 6.8 shows how our algorithm produces an office. While figure 6.9 show the generated living room. Figure 6.10 is an interesting result, because while the target room type is bedroom. The input room shape belongs to an Office which is part of our training set. This result shows that our EnvironmentFormer doesn't overfit on room shapes.

Despite the collision of the objects in the bedroom. These results prove that our novel approach to encoding the room type works. EnvironmentFormers don't need to be re-trained

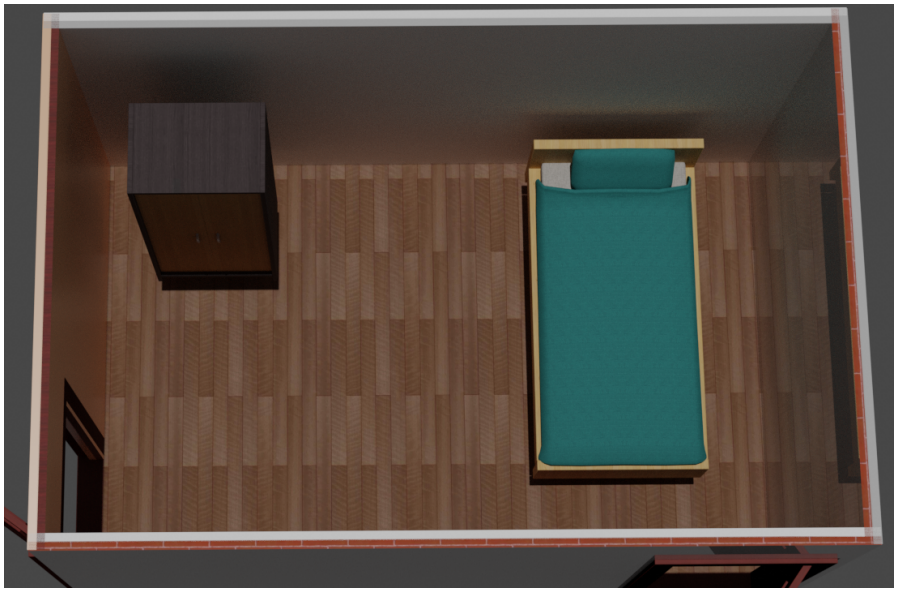


Figure 6.3: A bedroom generated by running the full algorithm as explained in section 4.7. The room has a bed, a stand and a wardrobe cabinet. A bedroom that was part of the training data, which has the same shape is shown in 6.4.

for each class of data, but rather can learn the class of which data goes through them by encoding the data class.

6.3 Styled rooms

In this section a styled room is shown and the result is explained.

In figure 6.12 is the style photo used to produce our results. The hyper parameters used for this result for the Neural Style transfer network. 100000 for the style weight was used and 10 for the content weight was used. These values could be tuned to change how much of weight of the original texture to have in the final one.

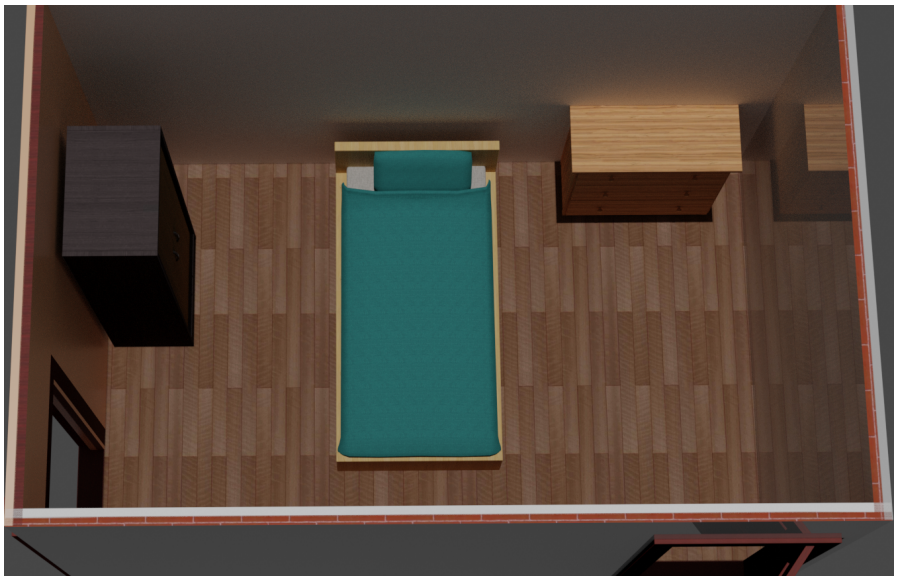


Figure 6.4: The closest bedroom in the dataset to the room generated in the figure 6.3

Figure 6.5: A bedroom generated by running the full algorithm as explained in section 4.7

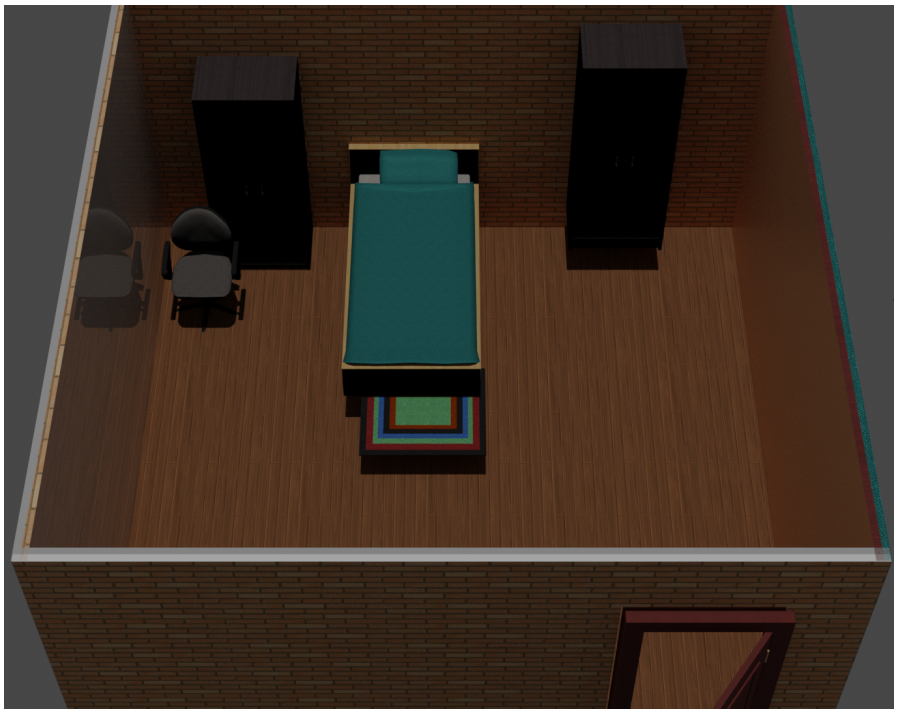


Figure 6.6: General structure of a bedroom still holds. The symmetry around the bed when there is a space still holds. Some limitation of the location EnvironmentFormer is shown in the position of the Chair. Not all the objects are in contact with the floor, some are floating like the rug and the wardrobe cabinet on the left of the bed.

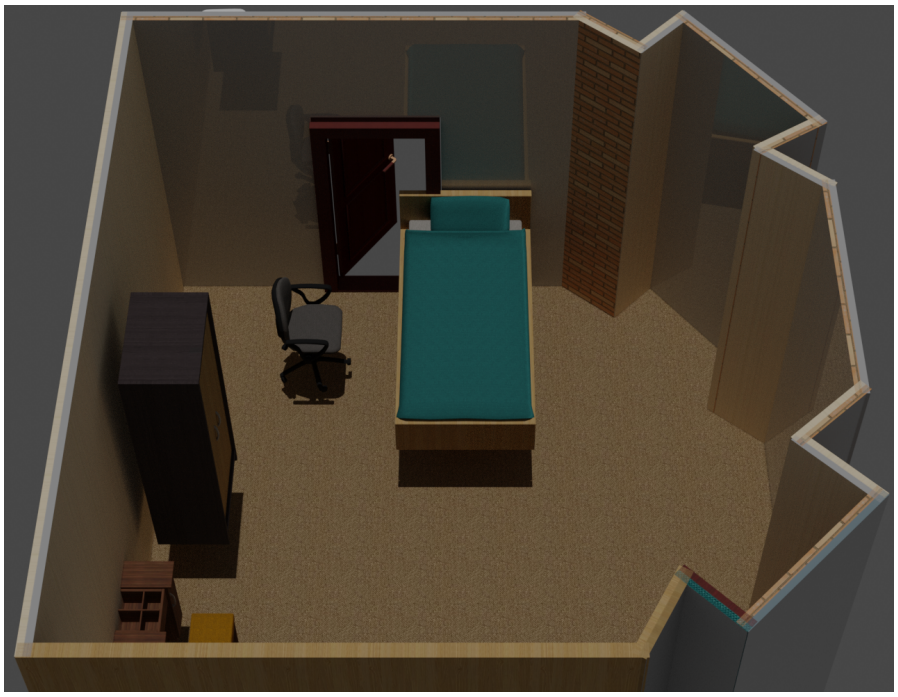


Figure 6.7: A bedroom generated by running the full algorithm as explained in section 4.7. This shows how our EnvironmentFormer works with non square or rectangle rooms.

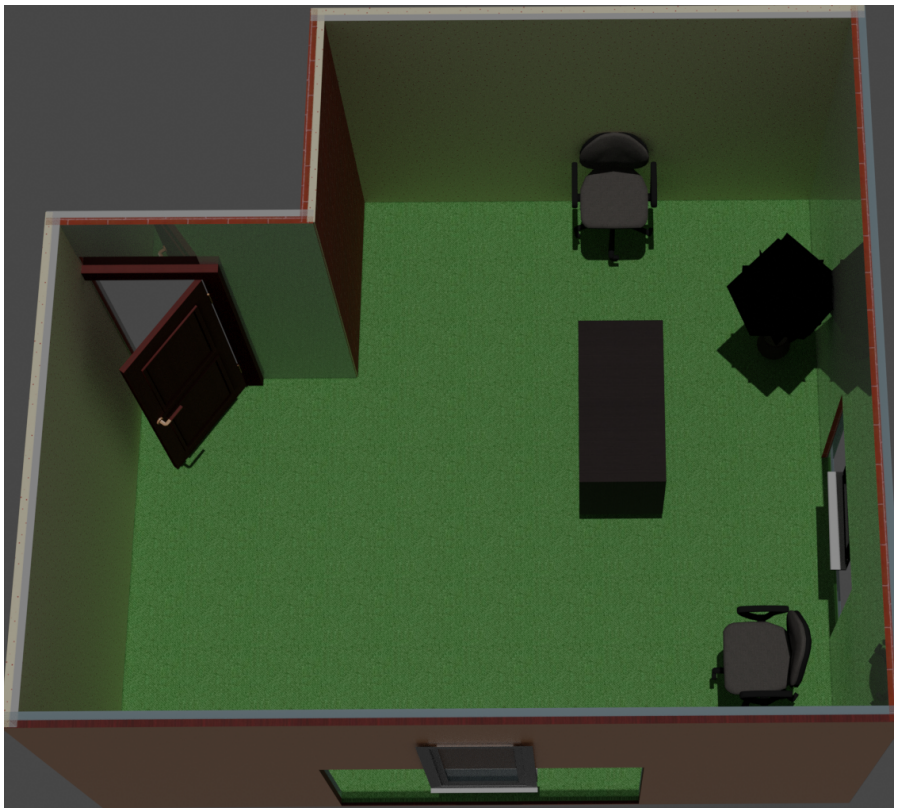


Figure 6.8: An office generated by running our algorithm with multiple room types.

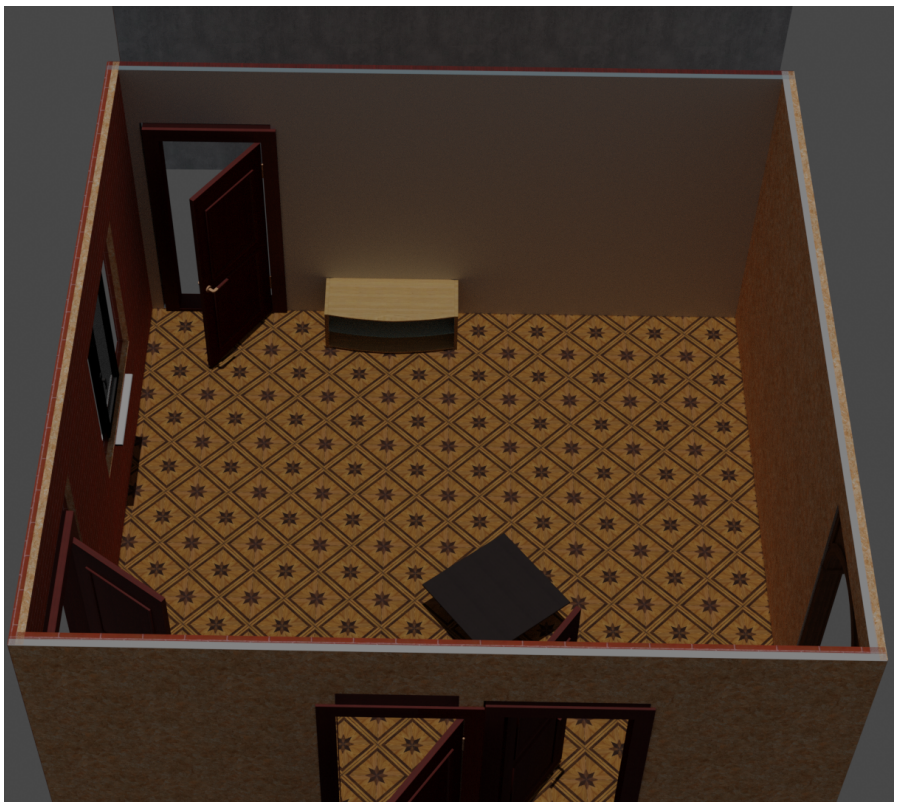


Figure 6.9: A living room generated by running our algorithm. When it came to the living room, the EnvironmentFormer predicts the stop token sooner than the other two room types. The conclusion that was drawn from this is that filtered living rooms need better augmentations.

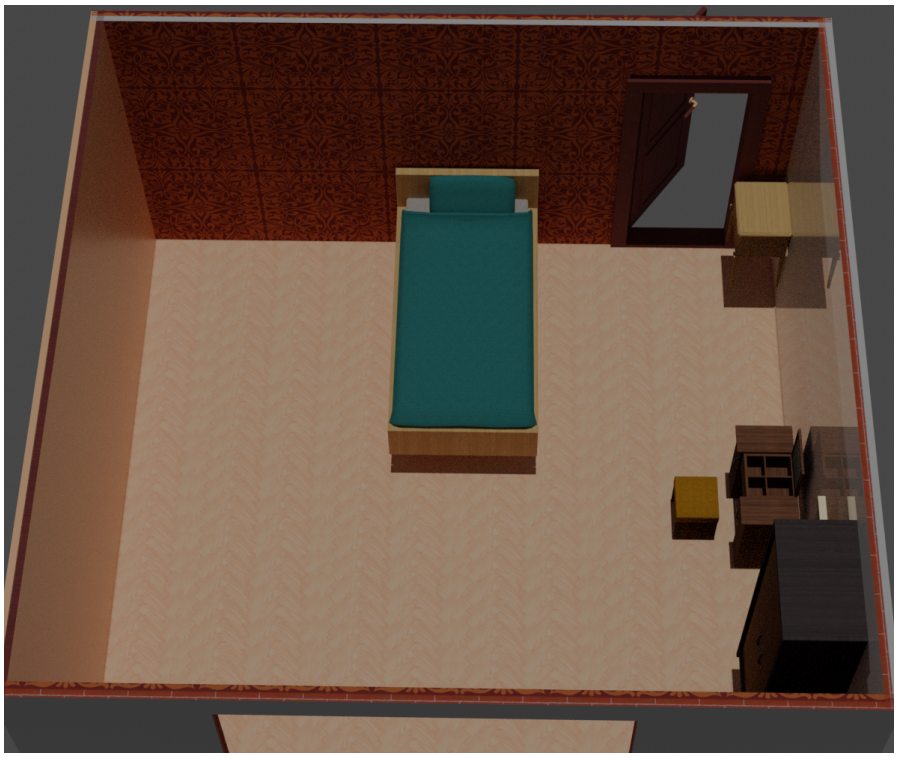


Figure 6.10: A bedroom generated by running the our algorithm. While the target is a bedroom , the input room shape belongs to an office in the training set.

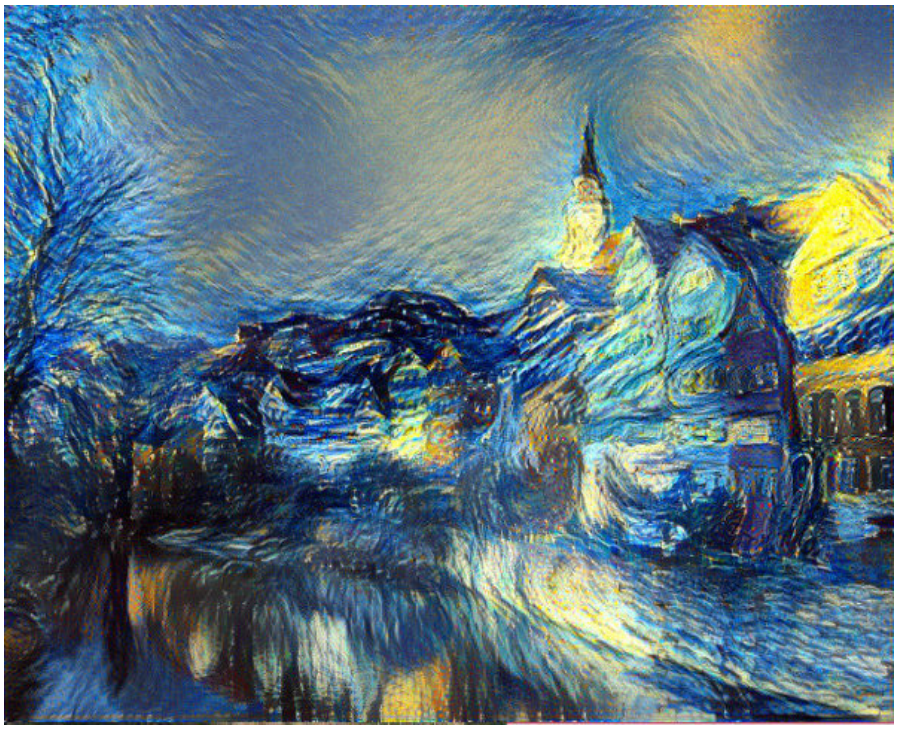


Figure 6.11: The style image used is the famous The Starry Night by Vincent van Gogh's .

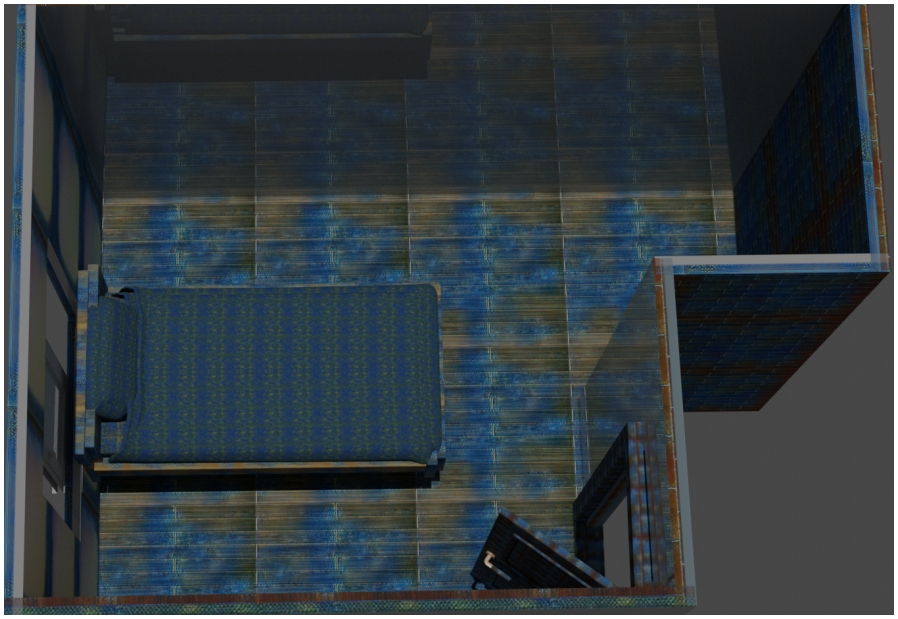


Figure 6.12: A bedroom generated and styled using our EnvironmentFormer.

7 Future Steps

In this chapter, the future steps to continue and improve this work is discussed.

7.0.1 Datasets and Data representation

More data, this means adapting this work to other indoor datasets. Front 3D [Fu+20] is a very good example for a dataset like this. Combining datasets results in better generalization. The increase in the amount of data would also result in a more better performance for EnvironmentFormers.

Applying EnvironmentFormers to another kinds of data is still an open question. Instead of training on indoor rooms, this EnvironmentFormers could be extended to generate other kinds of data. Any set of objects where the location and category are present, this EnvironmentFormers could work of them. For example, A chemicals factory floor is to be planned. This floor would should have machines, pipes, pathways for workers, computers and desks. With data for how a factory ground floors look like, and what kind of equipment and objects it has, and what kind of factory it is adopting EnvironmentFormers should in theory work.

Another future area of research for the EnvironmentFormers is, if it could learn more than one class of data at once. Meaning if it could learn how to design a whole apartment at once and place the rooms and connect them.

7.0.2 Room types Encoding strategies

In this thesis, our approach to this was to convert each room type to a different grey scale value. Encoding the information in the embedding of the room shape information.

Some other strategies could focus to solve this problem more from the transformer side. Adding a sequence dedicated for this information could be an option, or integrating the information into the already existent sequences.

7.0.3 Styling rooms

In this thesis, our approach was to style the textures with a style image, using neural style transfer. The Neural style transfer network result is affected heavily by the parameters it is given. More research is needed on how to style textures without the textures loosing their original looking material when styled.

8 Conclusion

In this thesis, two approaches to the task of Indoor 3D scene generation are presented. In the first approach, it is shown how a transformer [Vas+17] network could be used for Indoor 3D scene generation, treating it as a sequence generation task [WYN20].

In our second approach, EnvironmentFormer is introduced. Based on the transformer model, our novel design allowed for learning multiple room types in a single EnvironmentFormer. This solved the problem of having model weights for each room type the transformer is trained on. Saving time and space, and making the task of Indoor 3D scene generation more practical.

The EnvironmentFormer is not constrained by a room shape or a room type. A square room is not required like the work from [Wan+18] and most of the solutions that involved Convolutional Neural Networks.

List of Figures

List of Tables

Bibliography

[Com18] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018.

[Daw] Dawson-Haggerty et al. *trimesh*. Version 3.2.0.

[Fal19] W. Falcon. "PyTorch Lightning." In: *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* 3 (2019).

[Fu+20] H. Fu, B. Cai, L. Gao, L. Zhang, C. Li, Q. Zeng, C. Sun, Y. Fei, Y. Zheng, Y. Li, Y. Liu, P. Liu, L. Ma, L. Weng, X. Hu, X. Ma, Q. Qian, R. Jia, B. Zhao, and H. Zhang. "3D-FRONT: 3D Furnished Rooms with layOuts and semaNTics." In: *arXiv preprint arXiv:2011.09127* (2020).

[GEB15a] L. A. Gatys, A. S. Ecker, and M. Bethge. "A Neural Algorithm of Artistic Style." In: *CoRR* abs/1508.06576 (2015). arXiv: 1508.06576.

[GEB15b] L. A. Gatys, A. S. Ecker, and M. Bethge. "Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks." In: *CoRR* abs/1505.07376 (2015). arXiv: 1505.07376.

[He+16] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition." In: June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[JGS19] M. Jaritz, J. Gu, and H. Su. "Multi-view PointNet for 3D Scene Understanding." In: *CoRR* abs/1909.13603 (2019). arXiv: 1909.13603.

[Kha+19] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. "A Survey of the Recent Architectures of Deep Convolutional Neural Networks." In: *CoRR* abs/1901.06032 (2019). arXiv: 1901.06032.

[Lin+21] T. Lin, Y. Wang, X. Liu, and X. Qiu. "A Survey of Transformers." In: *CoRR* abs/2106.04554 (2021). arXiv: 2106.04554.

[Liu+14] T. Liu, S. Chaudhuri, V. G. Kim, Q.-X. Huang, N. J. Mitra, and T. Funkhouser. "Creating Consistent Scene Graphs Using a Probabilistic Grammar." In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 33.6 (Dec. 2014).

[Ma+17] L. Ma, J. Stückler, C. Kerl, and D. Cremers. "Multi-View Deep Learning for Consistent Semantic Mapping with RGB-D Cameras." In: *CoRR* abs/1703.08866 (2017). arXiv: 1703.08866.

[Nas+20] C. Nash, Y. Ganin, S. M. A. Eslami, and P. W. Battaglia. *PolyGen: An Autoregressive Generative Model of 3D Meshes*. 2020. arXiv: 2002.10880 [cs.GR].

[NF12] P. K. Nathan Silberman Derek Hoiem and R. Fergus. "Indoor Segmentation and Support Inference from RGBD Images." In: *ECCV*. 2012.

[NKP18] M. Naseer, S. H. Khan, and F. Porikli. "Indoor Scene Understanding in 2.5/3D: A Survey." In: *CoRR* abs/1803.03352 (2018). arXiv: 1803.03352.

[Pas+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035.

[RWL18] D. Ritchie, K. Wang, and Y.-A. Lin. "Fast and Flexible Indoor Scene Synthesis via Deep Convolutional Generative Models." In: *CoRR* abs/1811.12463 (2018). arXiv: 1811.12463.

[Son+17] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. "Semantic Scene Completion from a Single Depth Image." In: *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition* (2017).

[SZW19] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. "Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations." In: *CoRR* abs/1906.01618 (2019). arXiv: 1906.01618.

[Vas+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[Vid+21] M. Vidanapathirana, Q. Wu, Y. Furukawa, A. X. Chang, and M. Savva. "Plan2Scene: Converting Floorplans to 3D Scenes." In: *CoRR* abs/2106.05375 (2021). arXiv: 2106.05375.

[Wan+18] 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.

[Wan+19] K. Wang, Y.-A. 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.* 38.4 (2019), 132:1–132:15.

[WYN20] X. Wang, C. Yeshwanth, and M. Nießner. "SceneFormer: Indoor Scene Generation with Transformers." In: *arXiv preprint arXiv:2012.09793* (2020).

[Zho+18] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba. *Semantic Understanding of Scenes through the ADE20K Dataset*. 2018. arXiv: 1608.05442 [cs.CV].

[ZWK19] Y. Zhou, Z. While, and E. Kalogerakis. "SceneGraphNet: Neural Message Passing for 3D Indoor Scene Augmentation." In: *CoRR* abs/1907.11308 (2019). arXiv: 1907.11308.