3D Room Generation from 2D Semantic Layout

Jason Farnsworth
jfarnsworth@randolphcollege.edu
Randolph College
Lynchburg, Virginia, USA

Zitong Zhang
zitong.zhang@louisville.edu
University of Louisville
Louisville, Kentucky, USA

Rui Yu
rui.yu@louisville.edu
University of Louisville
Louisville, Kentucky, USA

Abstract

We propose a cascaded 3D diffusion pipeline for generating 3D indoor scenes using Truncated Signed Distance Functions (TSDFs) with 2D semantic layout as input. The pipeline is designed to run in a sparse occupancy space and in a coarse to fine fashion. The DiffInDScene [5] diffusion model is used as a base due to its ability to generate 3D indoor scenes using 2D sketches. Our proposed method would show an improvement in generating furniture within indoor scenes with the information provided by 2D semantic layouts.

CCS Concepts

• **Artificial Intelligence, Computer Vision**;

Keywords

3D Room Generation, Diffusion Model, 2D Semantic Input, Furniture Generation

1 Introduction

3D room generation is an important task in the field of Computer Vision, as it can be used in many different applications, such as video game development, augmented reality, film development, and construction [14] [15]. However, producing 3D room generations is a challenge, due to their diversity and complexity. Generated rooms tend to not adhere to typical room layouts, with additional furniture and inconsistent theming and geometry, as well as objects clipping into each other. Even ignoring these issues, the quality of generated meshes is far from perfect, with details being lost when the mesh is generated [5].

Many methods of 3D room generation have been proposed and created. One method involves an RGB picture of a room and determining the semantic lines (wall-wall, wall-floor, wall-ceiling) to guide the construction of the scene [9]. Another method involves using 3D semantic proxy rooms as input and reconstructing the scene using bounding boxes [10] [15] . Models have also utilized a text prompt feature, allowing the user to type out a desired style for the generated scene [3] [10] [16]. However, diffusion models are starting to be the method of choice to address the problem of room generation.

Diffusion models are being used in 3D room generation because they can circumvent the issues of inconsistent geometry and additional furniture due to their iterative denoising process. In our work, we build on the work of DiffInDScene [5], a framework for tackling the issue of high-quality 3D indoor scene generation. We implement the use of 2D semantic layouts to create 3D indoor room scenes. However, due to time constraints and model setup issues, we were unable to fully create our proposed pipeline.

2 Related Works

2.1 Room Generation

As stated in **Introduction**, there have been multiple methods used in the problem of room generation. These methods utilize real world images as their input [9] [14] [16] to aid in the generation of the scene. [9] and [14] produce a layout estimation as their output, showing the depth of the input room. [16] uses an image as its input, but it can also take in a sketch or text prompt describing the room. RoomCraft [16], unlike [9] and [14], creates realistic indoor scenes as opposed to layout estimations.

2.2 Diffusion Models

Diffusion models have been a popular method recently in room generation due to their iterative process. This is useful in cases such as shape denoising [2] [5] [15], adding geometry [3], and keeping theme consistency [10]. Text-based prompts are especially useful with diffusion models. Text2Room [3], ControlRoom [10], and SceneCraft [15] all utilize a text prompt as their input, while ControlRoom utilizes an additional 3D room layout built on semantic bounding boxes and SceneCraft uses semantic bounding boxes and a camera trajectory. DiffInDScene uses just a sketch to generate its scene, using diffusion to generate scenes in a "coarse-to-fine" fashion, and Frankenstein [2] uses a 2D semantic layout with diffusion being used in the noising and denoising process.

2.3 Object Generation

Object generation is useful in room generation to help generate objects, like furniture, clearly. Diffusion models are also used in object generation in a "coarse-to-fine" fashion [4] [8], with [4] using TSDFs and [8] using voxel grids. However, other methods such as generative adversarial networks [13] and cross attention grids [11] are used to generate 3D objects. Frankenstein [2] is capable of generating compositional avatars by generating body, clothing, and hair parts separately, allowing those parts to be connected manually.

3 Methodology

3.1 Theoretical Information

Our proposed method builds on DiffInDScene [5] to create our 3D room generations. We modify the code that processes input, update the SketchVAE model, upsample the latent space DiffInDScene works in, and retrain the SketchVAE model on a dataset with 2D semantic layouts and their corresponding 3D meshes. 2D semantic data would be used from the 3D-Front dataset [1]. Once the model is sufficiently trained on 2D semantic data and shows sufficient performance, furniture generation within the indoor scenes would be the next emphasis. DiffInDScene generates adequate furniture

in unconditional and sketch conditional generation, but is limited in the placement of furniture due to the lack of any meaningful information regarding furniture in the input to the model. The sketches the model takes in as input only contain information about the layout of the room, with no additional information about furniture objects. 2D semantics overcome this issue by providing location and size information about furniture present in the layout. A diagram for the pipeline of the proposed model is shown in Figure 1.

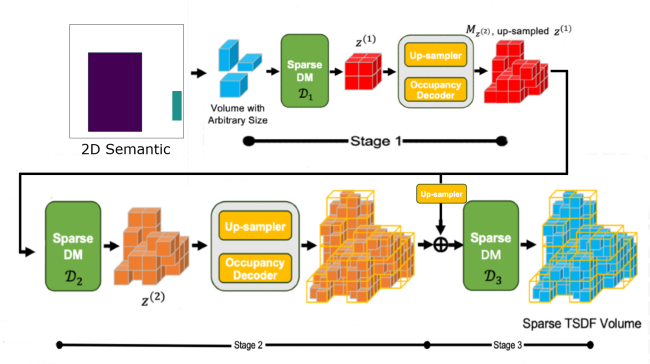

Figure 1: In stage 1, starting with a semantic layout and pure noise, the model generates a very coarse room structure influenced by the semantic using a sparse diffusion model. In stage 2, the output of stage 1 is ran through another sparse diffusion model to generate a finer version of the scene. Stage 3 takes the output of stage 2 and refines it using TSDFs through a third sparse diffusion model, using the coarse structure from stage 1 to stay consistent in its generation. Portions of this diagram came from DiffInDScene and were edited using Pixlr [6], an online photo editor.

However, multiple issues occurred that prohibited us from creating our proposed method. The first issue that arose was difficulty in downloading and setting up the DiffInDScene model. During the installation, the incorrect version of PyTorch was downloaded, which caused model performance issues later down the line. This error was not detected until the DiffInDScene pipeline had to be reinstalled on another workstation, and the correct version of PyTorch was installed when cloning with an Github SSH key instead of Github HTTPS cloning. The second issue was a result of using the incorrect PyTorch version, as numerous weeks were spent debugging the pipeline to work properly. In that time, we were able to get results for steps 1 and 2 of the testing part of the pipeline, but the results were very blocky and inaccurate. Additional weeks were spent debugging step 3 of the testing phase and trying to change steps 1 and 2 to generate better results, but those efforts were halted when the pipeline had to be moved.

Progress was made on converting the pipeline to support 2D semantic layouts, and those details will be covered in the **Implementation Details** section.

3.2 Implementation Details

Some progress was made on converting the pipeline to support 2D semantic layouts. The first thing that was changed was the code that handled the input and passed it downstream. Originally, the variable `sketch_file` pointed to the 2D input sketch, but now points to the 2D semantic input. Originally, the pipeline took in sketch inputs using `cv.imread(sketch_file, cv.IMREAD_GREYSCALE)`, where `cv.IMREAD_GREYSCALE` and `bev_sketch[bev_sketch > 0] = 1`, the binarization process, kept the sketch using black and white. We changed `cv.imread(sketch_file, cv.IMREAD_GREYSCALE)` to `cv.imread(sketch_file, cv.IMREAD_COLOR)` and removed binarization to preserve the colors in the semantic.

The above changes were made for the model to support 3-channel input (RGB), rather than just 1-channel input (greyscale). Tensors then had to be changed to support the 3-channel input. The variable `bev_semantic_layout_raw`, which was originally `bev_sketch`, was the result of the input after being read through `cv.IMREAD_COLOR`. Variables `bb_min` and `bb_max` were originally used to determine the bounding boxes of the active regions. However, the dimensions of `bb_min` and `bb_max` did not fit with the 3D latent space due to the width of `bb_min` and `bb_max` going far beyond the maximum width value for the 3D latent space, which was 16. The width for `bb_min` and `bb_max` had to be capped at 15 to not raise an `IndexError: index out of bounds` error. The spatial dimensions for the mask for the first level of the diffusion process, `code1_mask`, had to be changed to the correct dimensions for semantics. Both `code1_masks` corresponded to `(Batch, Mask_Channels, Depth, Height, Width)`, but the dimensions of `code1_mask` for sketches was `torch.Size([1, 4, 16, 64, 64])`, while the `code1_mask` for semantics had to be `torch.Size([1, 1, 64, 64, 16])`.

Once the input and tensors were changed to support 2D semantics, the `SketchVAE` portion of the model had to be changed. The `SketchVAE` is responsible for encoding the input sketch into a compact, lower-dimensional 3D latent representation. The main things that were changed about the `SketchVAE` was the expected in and out channels, which were changed from 1 to 3, and the disabled support of the given checkpoints for the model, which were created with the sketch information in mind. However, there were issues with disabling the support of the previous checkpoints, as the model still pulled information from them even when using `strict=False` in the `load_state_dict` function call. This resulted in the error `The size of tensor a (256) must match the size of tensor b (1024) at non-singleton dimension 3`, which we were unable to fix in the time allotted for the project.

4 Experiments

We were not able to run any tests on our proposed method due to not being able to implement our proposed method. Instead, experiments were run on the DiffInDScene [5] model to replicate results and determine the model's limitations. The unconditional generation and sketch conditioned generation parts of the model were tested, and their results are discussed in the corresponding sections. Meshes were viewed and colored in Blender [12].

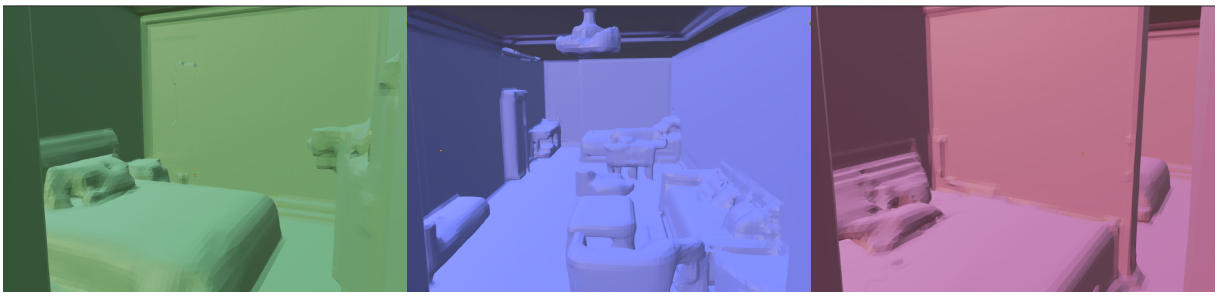

Figure 2: Perspectives of multiple rooms of the interior of the generated scene. All three rooms contain beds, due to the model estimating what furniture should be in each room.

4.1 Unconditional Generation

Unconditional generation was tested first as a way to make sure the model was installed and setup correctly. Figure 2 shows multiple interior perspectives, while Figure 3 shows the exterior of the generated scene.

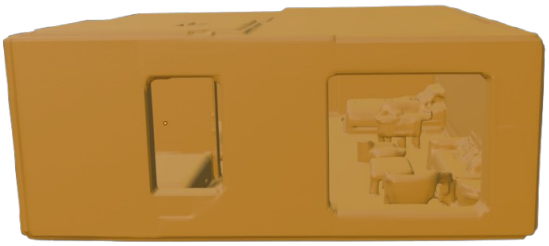

Figure 3: A front facing view of the unconditionally generated mesh. Only one side has open areas that allow viewing of the interior.

For unconditional generation, there are not a lot of opportunities to change the result mesh because the model creates a mesh from pure noise with no defined condition. This fact is apparent in the context of furniture generation, as the model estimates what furniture should go in each room. Model checkpoints from DiffInDScene were used in unconditional generation.

4.2 Sketch Conditioned Generation

For sketch conditioned generation, we have more control over what the model generates with input guiding the structure and look of the final result. Figure 4 shows some of the sketches used as input and their results.

Layout 1 was the sketch that was a native part of DiffInDScene, and as such, performed the best in its generation. Layout 2 performed the worst, with the general structure of the generated mesh not matching the input layout. Layouts 3 and 4 were created to mimic semantic layouts, as opposed to sketched layouts, and their generated meshes structurally look similar to their input. Layout 3 uses rectangles to mimic bounding boxes for furniture in actual semantic layouts, and layout 4 uses more detailed furniture icons. Furniture was added in the input to determine if the model could

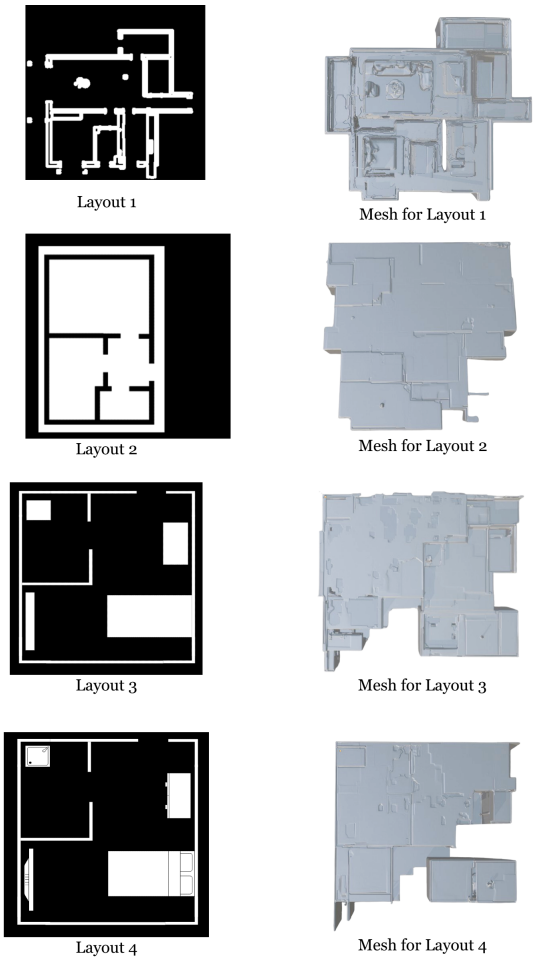

Figure 4: Layout 1 was the original sketch used by DiffInDScene, while we created the other layouts. Our layouts were created using Visual Paradigm Online [7], an online tool used to create floor plans.

differentiate between furniture and wall structure. Unfortunately, but as expected, the model interpreted the furniture in the layouts

Figure 5: Close up picture of the room the model created instead of the bed in layout 4. A bed is also placed inside this room, showing that the model is predicting what furniture should be placed and that furniture generation is not directly controllable.

as wall structures and created mini rooms in the locations of the furniture. Figure 5 shows a close up shot of the room the model created where the bed should have been in layout 4.

Something to note is that these tests were run on the "off the shelf" version of DiffInDScene, which was built with pure sketches as the intended input. When running sketches that resembled semantics, like layouts 3 and 4, this caused the model to "break" and perform worse on subsequent inputs. Figure 6 shows an example of a "broken" output.

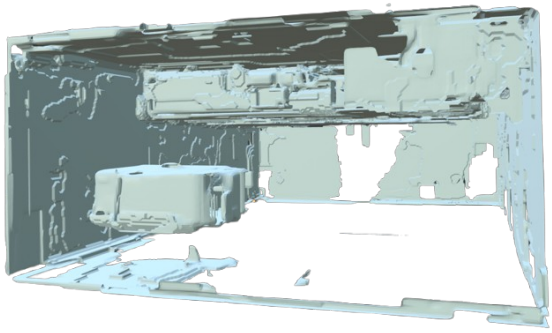

Figure 6: A result mesh after being ran through the model after a semantic. It is "broken" due to how open the inside of the mesh is, as well as how thin and decayed the walls are.

Detailed perspectives of the meshes in Figure 4 can be found in the Appendix.

5 Future Work

As mentioned in the **Implementation Details** section, some progress was made on converting the pipeline to support 2D semantic layouts. However, progress was halted on editing the SketchVAE part of the model due to time constraints. In the future, we hope to overcome this issue, as well as retraining the model and running experiments. The dataset we would use to retrain the SketchVAE would be 3D-Front [1], the same dataset the SketchVAE was originally trained on. However, the SketchVAE was only trained on 3D indoor scenes, whereas we would train it on 2D semantics in conjunction with 3D indoor scenes. We also want to explore the potential of DiffInDScene's [5] furniture generation, finding a way to implement better control of where furniture is generated and what types of furniture are generated. This would ideally be tackled through the use of semantics over sketches as input.

6 Conclusion

In this paper, we propose a cascaded 3D diffusion pipeline for generating 3D indoor scenes using TSDFs with 2D semantic layout as input. However, due to issues and time constraints, we were unable to fully create our proposed method. We show results from DiffInDScene [5], testing the limits of its indoor scene generation, as well as discussing the work that was completed and the work that will be completed to fully create our proposed method for generating 3D indoor scenes with 2D semantic layouts as input.

References

[1] Huan Fu et al. "3D-FRONT: 3D Furnished Rooms with layOuts and semaNTics". In: *CoRR* abs/2011.09127 (2020). arXiv: 2011.09127. URL: https://arxiv.org/abs/2011.09127.

[2] Yan Han et al. "Frankenstein: Generating Semantic-Compositional 3D Scenes in One Tri-Plane". In: *ACM SIGGRAPH Asia Conference Proceedings* (2024).

[3] Lukas Höllein et al. *Text2Room: Extracting Textured 3D Meshes from 2D Text-to-Image Models*. 2023. arXiv: 2303.11989 [cs.CV]. URL: https://arxiv.org/abs/2303.11989.

[4] Ka-Hei Hui et al. *Neural Wavelet-domain Diffusion for 3D Shape Generation*. 2022. arXiv: 2209.08725 [cs.CV]. URL: https://arxiv.org/abs/2209.08725.

[5] Xiaoliang Ju et al. "DiffInDScene: Diffusion-based High-Quality 3D Indoor Scene Generation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 4526–4535.

[6] Pixlr Pte Ltd. *Pixlr Photo Editor*. Accessed: 2025-07-17. 2025. URL: https://pixlr.com/.

[7] Visual Paradigm Online. *Visual Paradigm Online Homepage*. Accessed: 2025-07-15. 2025. URL: https://online.visual-paradigm.com/.

[8] Xuanchi Ren et al. "XCube: Large-Scale 3D Generative Modeling using Sparse Voxel Hierarchies". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024.

[9] Denys Rozumnyi et al. *Estimating Generic 3D Room Structures from 2D Annotations*. 2023. arXiv: 2306.09077 [cs.CV]. URL: https://arxiv.org/abs/2306.09077.

[10] Jonas Schult et al. *ControlRoom3D: Room Generation using Semantic Proxy Rooms*. 2023. arXiv: 2312.05208 [cs.CV]. URL: https://arxiv.org/abs/2312.05208.

[11] Etai Sella et al. *Vox-E: Text-guided Voxel Editing of 3D Objects*. 2023. arXiv: 2303.12048 [cs.CV]. URL: https://arxiv.org/abs/2303.12048.

[12] Blender Development Team. *Blender*. Accessed: 2025-07-17. 2025. URL: https://www.blender.org/.

[13] Jiajun Wu et al. *Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling*. 2017. arXiv: 1610.07584 [cs.CV]. URL: https://arxiv.org/abs/1610.07584.

[14] Chenggang Yan et al. "3D Room Layout Estimation From a Single RGB Image". In: *IEEE Transactions on Multimedia* 22.11 (2020), pp. 3014–3024. DOI: 10.1109/TMM.2020.2967645.

[15] Xiuyu Yang et al. *SceneCraft: Layout-Guided 3D Scene Generation*. 2025. arXiv: 2410.09049 [cs.CV]. URL: https://arxiv.org/abs/2410.09049.

[16] Mengqi Zhou et al. *RoomCraft: Controllable and Complete 3D Indoor Scene Generation*. 2025. arXiv: 2506.22291 [cs.CV]. URL: https://arxiv.org/abs/2506.22291.

A In-Depth Look at Sketch Generated Meshes

This section takes a look at multiple views of the meshes from figure 4 and analyzes what the model created based on the sketch provided.

A.1 Layout 1

Figure 7 shows the location for the images shown in **Layout 1**.

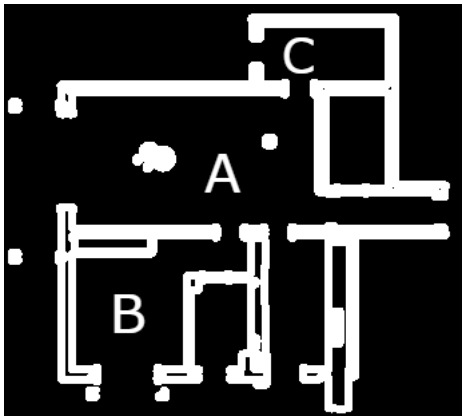

Figure 7: A detailed key for layout 1. Label *A* represents the location of the first image, *B* represents the location of the second image, and *C* represents the location of the third image.

Figure 8: A picture of the mesh of area *A*. The model interpreted the area as a living room, and added objects like a couch and chairs.

Figure 9: A picture of the mesh of area *B*. The model interpreted the area as a bedroom and placed a single bed in the room.

Figure 10: A picture of the mesh of area *C*. The model interpreted the area as a spare area and has placed a bench-like object in the room.

A.2 Layout 2

Figure 11 shows the location for the images shown in **Layout 2**.

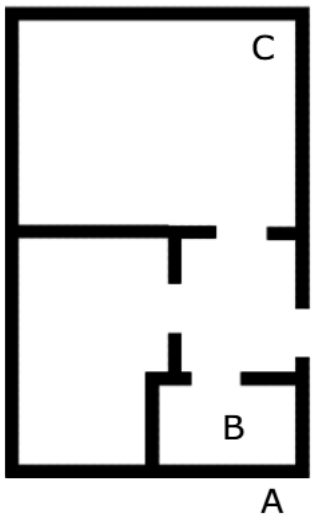

Figure 11: A detailed key for layout 2. Label *A* represents the location of the first image, being the wall itself, *B* represents the location of the second image, and *C* represents the location of the third image.

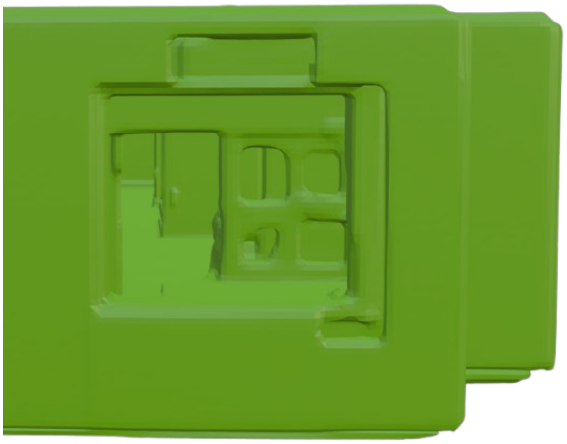

Figure 12: A picture of the mesh of area *A*. The model attempted to add a window for the open space, but only generated it for half of the space.

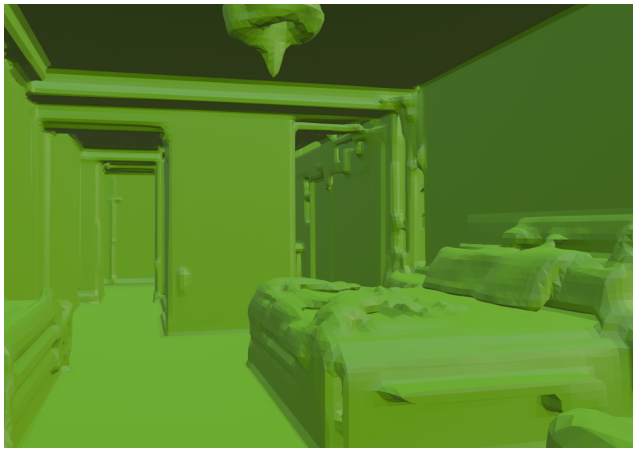

Figure 13: A picture of the mesh of area *B*. The model interpreted the area as a bedroom, but has added more furniture in the room.

Figure 14: A picture of the mesh of area *C*. The model also interpreted the area as a bedroom, and has even enclosed the area despite area C being open and wide.

A.3 Layout 3

Figure 15 shows the location for the images shown in **Layout 3**.

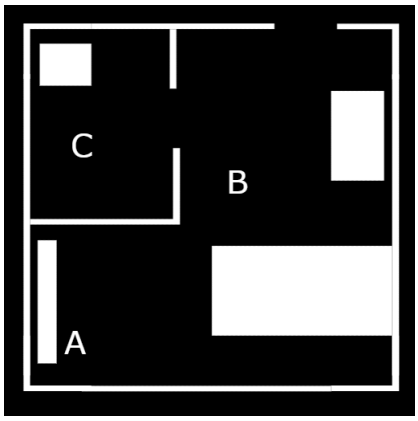

Figure 15: A detailed key for layout 3.

Figure 16: A picture of the mesh of area *A*. The rectangle is meant to represent a television, but the model has interpreted the area to be its own room. The model may have interpreted the area as a bathroom, with a sink and bathtub as potential furniture.

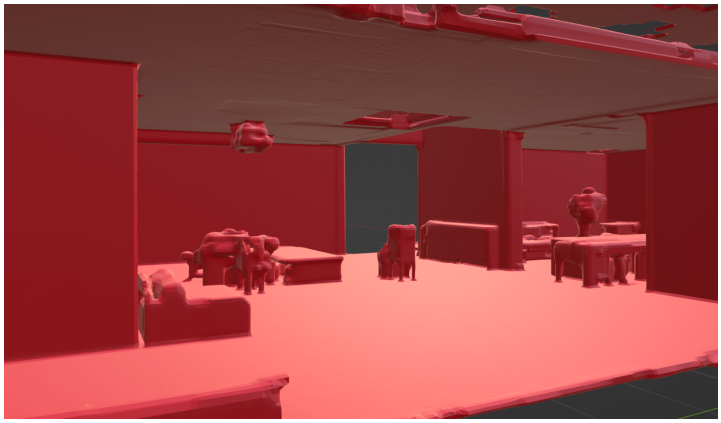

Figure 17: A picture of the mesh of area *B*. The model has interpreted the area as an open living space, which matches the sketch for area B.

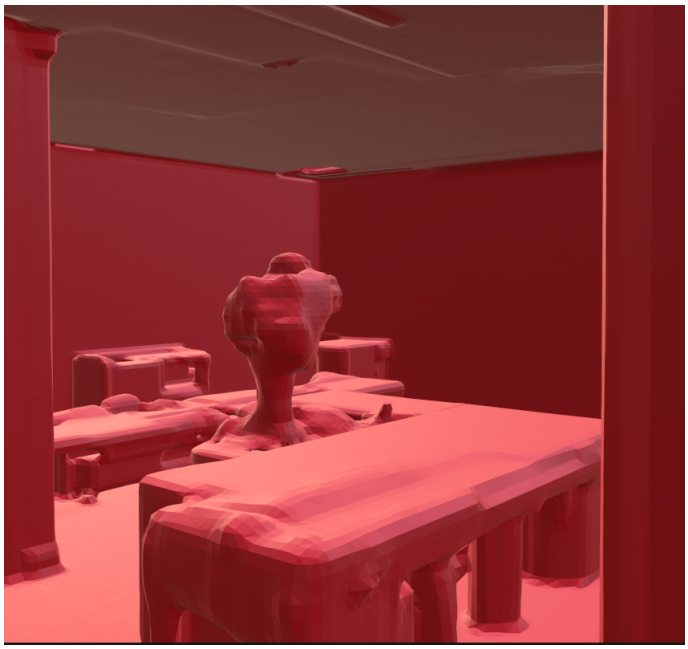

Figure 18: A picture of the mesh of area *C*. The area was intended to represent a bathroom, but the model has interpreted the area as another open living space, possibly being a continuation of area B.

A.4 Layout 4

Figure 19 shows the location for the images shown in **Layout 4**. Layout 4 is very similar to layout 3, but layout 4 has more detailed furniture in its layout.

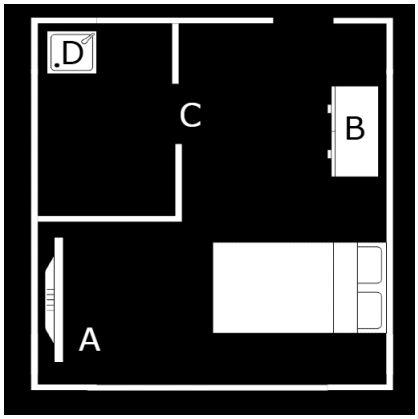

Figure 19: A detailed key for layout 4.

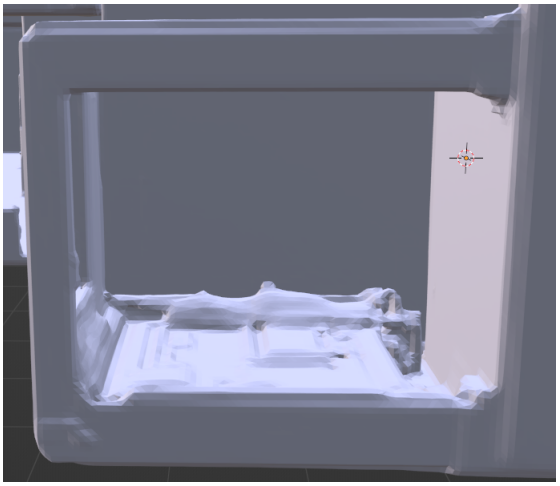

Figure 21: A picture of the mesh of area *B*. The shape was intended to be a dresser, but the model has interpreted the area as a very small bedroom.

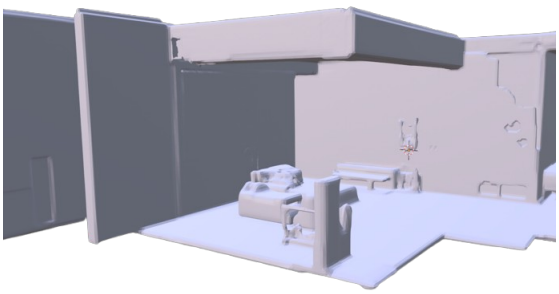

Figure 20: A picture of the mesh of area *A*. Unlike in layout 3, the model has interpreted the area as a bedroom, a common pattern.

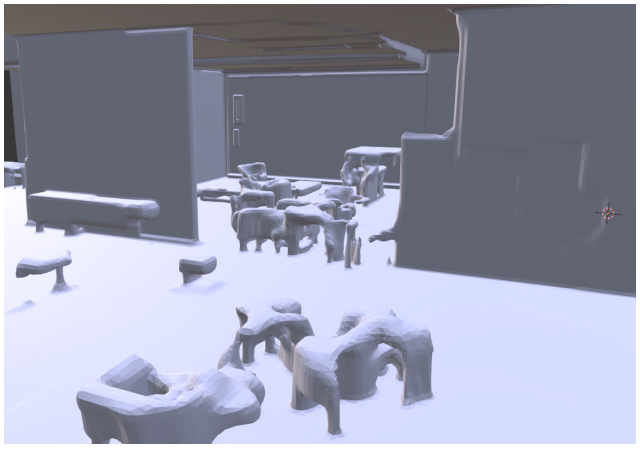

Figure 22: A picture of the mesh of area *C*. The model interpreted the area as an open living space, similar to its interpretation in layout 3.

Figure 23: A picture of the mesh of area *D*. The shape was intended to be a shower, and the model has created an enclosed area similar to that of a shower or closet.