ETH zürich

Computer Vision and Geometry Lab

AR Environment Design with Stable Diffusion

Semester Project Report

Mert Ertugrul (21-950-795)

Supervisor: Prof. Dr. M. Pollefeys
Advisors: Iro Armeni
Silvan Weder

30.6.2023

Contents

1 Introduction **2**

2 Related Work **3**
2.1 2D Generative Models . . . 3
2.2 3D Generative Models and Systems . . . 3
2.2.1 Text to Shape . . . 3
2.2.2 Text to Scene . . . 4

3 Models and Methods **4**
3.1 Pipeline Overview . . . 4
3.2 Server Side . . . 5
3.2.1 Image Generation . . . 5
3.2.2 Mesh Generation . . . 8
3.3 Client Side . . . 9

4 Experiments & Results **10**
4.1 Depth Evaluation . . . 11
4.1.1 Effect of the Number of Iterations on Depth Maps . . . 12
4.1.2 Effect of the Condition Type on Depth Maps . . . 12
4.2 2D and 3D Evaluation . . . 12
4.2.1 2D RGB Image and 3D Results . . . 13
4.3 2D User Evaluation . . . 15
4.4 2D User Evaluation Results . . . 15
4.5 Final Client Application Recordings . . . 18

5 Discussion & Conclusion **18**
5.1 Discussion of Evaluation Results . . . 18
5.2 Issues and Shortcomings . . . 18
5.3 Future Work . . . 19

A Appendix **22**
A.1 System Implementation Codebase . . . 22

A.2 Supplementary Client Application Screenshots 22

1 Introduction

Augmented reality (AR) has emerged as a powerful technology, enabling the overlay of virtual content onto the real world, thereby enhancing user experiences and opening up new possibilities in various domains. Creating AR experiences often involves designing virtual objects or scenes that seamlessly blend with the physical environment. This process usually relies on manual 3D modeling which may be time-consuming and require expertise in relevant tools. However, recent advancements in deep learning, particularly in the field of image, video, point cloud, and mesh generation, offer a promising avenue for automating the creation of augmented reality environments. Particularly, diffusion-based models have gained significant attention in the field of generative modeling due to their ability to generate high-quality images from complex probability distributions.

The idea of using diffusion based generative models for automatic augmented reality environment generation stems from the desire to streamline the content creation process and enable rapid prototyping of AR applications. By leveraging diffusion models in conjunction with text prompt models, we can generate images that align with specific textual descriptions or conditions. For example, given a text prompt describing a tranquil forest scene, a generative diffusion model can generate an image that represents the desired environment. Then, this image can be used to create a 3D representation. By combining the generated image and a corresponding depth map for the image inferred by monocular depth estimation, we can construct a point cloud representation of the generated scene. This point cloud can then be used to create a mesh, which can then be placed as a virtual object in an AR experience. To further enhance the quality of the generated environment and its adherence to the surrounding real environment, we utilize depth and segmentation maps as additional conditions, using variants of a pretrained conditioned generative diffusion pipeline, ControlNet[1]. By conditioning the diffusion model on these maps, we can control the spatial layout and object placement in the generated scene. This ensures that virtual objects are seamlessly integrated into the real-world environment, enhancing the overall AR experience.

This project aims to deliver a client-server system for Android phone clients that produces generated 3D environments on the server side and deploys the environments on the client side. We leverage a diffusion-based RGB image generation model, Stable Diffusion conditioned by geometric cues (depth map) and/or semantic cues using the ControlNet pipeline and produce a server-side pipeline for environment generation. The pipeline receives an RGB image, a phone-based initial depth map estimate, and a text prompt given by the user, and returns a mesh for the client app to render. The ultimate goal is to produce environments that are visually appealing, structurally accurate, placed at accurate coordinates while minimizing the waiting time of the user to approach a "real-time" experience as much as possible. This introduces a trade-off between mesh production time and visual quality as well as trade-off between geometric (depth) fidelity and visual appeal which emerges as an artifact of the conditioned generation pipeline.

Considering these constraints as well as the limitations of the client-side application development framework and its augmented reality capabilities, we implement an end-to-end system and document the impact of each stage of the pipeline on the quality of the final product.

2 Related Work

2.1 2D Generative Models

Text-to-image generative models have made leaps in recent years, beginning with OpenAI's DALL-E[2] in early 2021, which had the approach of combining a transformer with a variational autoencoder. Later, diffusion based generative models emerged as a groundbreaking new approach to generative modeling, following their proposal in the GLIDE paper[3]. Diffusion models apply consecutive steps of Gaussian noise to an initial image and train a model to predict the noise added at each step, where in the case of text-to-image models, a UNet model is used for noise prediction at each step. Then, new images are generated by sampling from the noise distribution. Subsequent models such as DALL-E-2[4], Imagen[5], and Stable Diffusion[6] built upon GLIDE, achieving further performance improvements by incorporating CLIP image embeddings, introducing classifier-free guidance, and leveraging large language models for text encoding, where Stable Diffusion and Dall-E22 operate on a latent space, which was observed to outperform operating on high dimensional pixel space [7]. The impressive level of photorealism and content diversity achieved by these models have prompted efforts to bring this generative capability to the 3D realm.

2.2 3D Generative Models and Systems

Being inspired by the success of diffusion models in 2D image generation, a variety of models and methods have recently emerged to either incorporate 2D generative models into pipelines for creating 3D environments or directly propose 3D point cloud or mesh generating diffusion models.

There is evident motivation behind circumventing the training of end-to-end text-to-3d models. The training of text-to-image models is a daunting task by itself given the immense size of the datasets, computing power and time required to do so. Given that 3D objects or scenes are significantly more complex than 2D images, we expect that a single diffusion model generating an arbitrary object or environment would require immense resources to produce, and scarcity of paired text-shape data for this purpose hampers development.

2.2.1 Text to Shape

Thus, a number of recent systems have leveraged 2D text-to-image models for optimizing parametric 3D volume or surface representations instead, such as DreamFusion[8], and DreamField[9] which optimize volumetric 3D models (NeRF) using gradient descent and SDFusion[10] which similarly optimizes a Signed Distance Field. Other approaches such as CLIP-Mesh decided against such parametric 3D representations and instead used a combination of Loop subdivision surfaces, texture maps, and normal maps for their representation, following a representation approach used in video games [11]. Finally, another recent system, ISS++[12], approaches the task of text-to-shape generation as a single view reconstruction (SVR) task, uses a diffusion prior for its generation, and uses CLIP for guiding the process.

It is important to note that the time required for generation for all of the above-mentioned systems is in the range of at least 30-90 minutes for stylized meshes. Their contributions are relevant to our purpose, but the aim of real-time generation does not fit such approaches.

As we aim to generate larger 3D environments that surround the user experiencing them, rather than individual objects that are meant to be viewed from all angles, it is also informative to inspect recent scene generation methods.

2.2.2 Text to Scene

The recent SceneScape[13] and Text2Room[14] approaches align remarkably well with our goals. SceneScape proposes an online method for text-driven perceptual view generation, synthesizing videos of scene walkthroughs using text prompts and camera poses. They combine a text-to-image model with geometric priors provided by a monocular depth estimation model. Their system is a zero-shot/test-time scene generation method, generating a scene frame by frame by inpainting using previously generated frames and iteratively optimizing a unified mesh as new frames are added.

Text2Room, on the other hand, proposes a method for generating room-scale textured 3D meshes. Their pipeline begins by using a 2D text-to-image model to synthesize a sequence of images from different poses and creates a consistent 3D scene using monocular depth estimation and text-conditioned inpainting. They propose a viewpoint selection strategy to maximize the structural cohesiveness of the room mesh, automatically engineer prompts according to which section of the room the viewpoint si directed at and as the pipeline iteratively renders a scene, they fuse new sections into the mesh.

3 Models and Methods

3.1 Pipeline Overview

In light of the existing works on scene generation, and keeping user waiting time as the primary constraint, we choose a simple approach that relies on a diffusion-based text-to-image model, Stable Diffusion, and depth cues from a monocular depth estimation model, MiDaS for single view mesh generation. We then build our client and server system to serve such a single-view AR experience, with the possibility to extend for a more advanced inpainting-based multi-view approach in the future. The majority of compute-heavy operations, including detailed depth map estimation, image generation, mesh construction and processing occur on the server side both for improved speed and for reducing the hardware burden on the client hardware. The client retrieves the data required for the server's operations and displays the end product. The responsibilities of both sides are explained in the relevant sections below.

A step-by-step overview of the system would play out as given below. Please refer to the system diagram in Figure 1 for a visual representation of the system. The steps are explained in further detail in the sections following the overview.

Client Side

1. Check that the server is available

2. Application UI becomes available. The user enters the desired text prompt in the relevant field. The user moves their device around the desired target section of their environment

for the environment to be registered. The user initiates the generation process by tapping the relevant button.

3. An RGB Image is captured by the rear camera and a corresponding coarse depth image is captured by ARCore's Depth API. The two images, the text prompt and camera intrinsics are sent to the server.

Server Side

4. Monocular depth estimation is carried out on the received RGB image. The produced depth map is relative. We store a version that is registered to the depth map received from the ARCore client for quality evaluation.

5. The relative (unregistered) depth map and text prompt are given to COntrolNetv1 with Stabel Diffusion 1.4 backbone. Inference is carried out for 10 steps, and a generated RGB image is produced.

6. Monocular depth estimation is carried out on the generated RGB image. The resulting relative depth map is registered to the ground truth depth map received from the ARCore client.

7. The generated RGB image, registered depth map of this image and camera intrinsics are used to create a point cloud.

8. The point cloud is preprocessed, the mesh generation algorithm is applied. The mesh is further processed. See Section 3.2.2 for a detailed explanation.

9. Files describing the mesh are produced, their URLs are sent to the client.

Client Side

10. The client requests and downloads the necessary mesh files. The client app constructs and places the generated mesh in the environment.

3.2 Server Side

The most significant sections of our pipeline are handled within our server. For the purposes of a proof of concept, our server is a simple Flask web server, having access to one NVIDIA GTX 1050 Ti GPU and 6 cores at 2.20GHz.

3.2.1 Image Generation

The image generation section of the pipeline has to consider that generated images are intended to be used for constructing 3D environments. In our pipeline, we make the conversion from generated 2D RGB image to 3D mesh using monocular depth estimates on the generated 2D image. Thus, this image must preserve certain visual patterns that exist in the original RGB image and have sufficient visual information to allow for a monocular depth model to extract a useful depth map matching the original geometry. One way of achieving the retention of such structural and/or semantic information in the generated image is to condition the generation process using

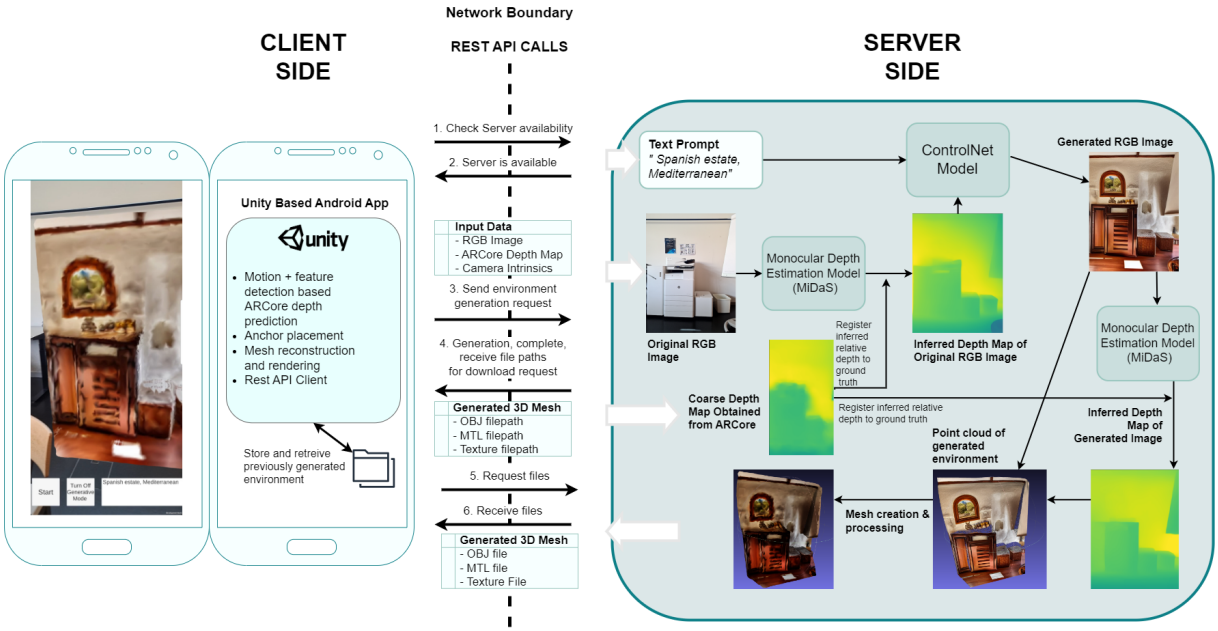


Figure 1 An overview of the final end-to-end pipeline including the Android client application and the server

depth maps, segmentation maps, or other types of cues such as normal maps, sketches, and line maps. Currently, the most widely preferred method of achieving such conditioning is via the ControlNet pipeline[1].

ControlNet ControlNet is an end-to-end neural network architecture that controls large image diffusion models, enabling task-specific input conditions. It consists of a "trainable copy" and a "locked copy" of weights from a diffusion model, connected by a unique "zero convolution" layer. The locked copy retains the capabilities of the original pretrained model, which is crucial given the vast amount of data and resources it takes to produce the given pretrained model. Meanwhile, the trainable copy is trained on task-specific data. Variants of ControlNet that use Stable Diffusion as their backbone where their trainable copies were trained on a variety of popular conditioning types are freely available. We opt for these variants as we intend for our pipeline to use open-source models and pipelines.

For the specific task of single view AR environment generation, three conditioning variants are proposed and evaluated for the pipeline, depth map, segmentation map, and a combination of both.

Depth Given that the end product of the pipeline is a mesh that must be overlaid onto existing surfaces in the real world through an AR experience, we aim to infer high-fidelity depth from the generated image such that it can be overlaid in an immersive manner. Therefore, conditioning generation on depth is an intuitive way of enforcing the preservation of attributes in the image that will lead to a successfully generated depth map that will have a similar structure to the ground truth depth.

Segmentation On the other hand, segmentation map conditioning offers a different type of information signal that may still be relevant. While depth fidelity is not directly enforced, segmentation maps can guide generation by specifying the types of objects that exist in the scene and lead to generated scenes that place objects of similar shape, size and category in the

scene. There are two primary issues with this approach. Segmentation maps color regions of the image into corresponding semantic categories, so the signal they provide can at most be an outline within the generated image. This does not include any geometric information regarding what is inside this outline. A large wall region may have originally had indentations, and protrusions, even a corridor. As long as these sections are all wall sections, they may be a part of the same continuous wall segment in the map and the corresponding geometry would be lost.

Multi-Condition The possiblity to condition generation on both map types simultaneously is possible in the ControlNet system. This has the potential of combining the useful information extracted from both maps. However, depending on the implementation of the conditioning and the way in which the conditions are combined, it may introduce excessively heavy conditioning as well, hampering visual quality, requiring more inference steps, or enhancing the shortcomings introduced by the individual conditions. Our evaluation of the different conditioning alternatives aims to determine where the currently available implementation of the ControlNet stands in its ability to leverage these conditions.

By conditioning the diffusion model on these maps, we can control the spatial layout and object placement in the generated scene. This ensures that virtual objects are seamlessly integrated into the real-world environment, enhancing the overall AR experience and quality of generated meshes.

Condition Input Preparation The server receives an original 2D RGB image as well as a depth map estimate of this image created by a client side stereo matching algorithm explained in Section 3.3. These inputs are the basis of conditioned image generation. We infer the condition maps to be used by the ControlNet model. When a depth map is used for conditioning, monocular depth estimation using the MiDaS model[15] is applied to the original RGB image. The resulting depth map is inverted and relative, and available pretrained ControlNet variants accepting depth maps are trained on inverted relative depth maps as well, thus the depth map is used as is. However, for later sections where depth maps are compared to each other, all depth maps are registered to the original depth map provided by the client side. The reason why this client side map is not used as the condition image is that it is coarse and does not preserve high-frequency details, previous trials using purely the client-side depth map produced significantly worse details and a different approach was taken. It is therefore used as a reference for the evaluation and registration of inferred maps. The same registration process that is used by the MiDaS implementation is applied[15], which begins by inverting the ground truth or reference depth map, followed by aligning the target depth map to the ground truth depth map by finding weight and offset values that minimize the least squares error between inverted reference and target depth maps and applying the weight and offset to the target depth map. Finally, the rescaled and shifted target depth map is inverted, as it is initially provided in inverted form and we need an uninverted version.

If a segmentation map is to be provided as a condition to ControlNet, the MaskFormer model[16] with a Swin transformer backbone[17] pretrained on ADE20k semantic segmentation dataset[18] since the available pretrained ControlNet variant using segmentation maps is also pretrained with the segmentation labels and color mapping corresponding to the ADE20k format.

Once either or both of the above mentioned condition maps are ready, we carry out inference with the pre-trained ControlNet pipeline. The number of inference steps applied at the stage as well as the image resolution used were both design considerations that had to be investigated. Lowering the resolution to values below roughly 384 pixels for the shorter dimension of the input

image was detrimental to generation quality, thus the image resolution was kept at this value for images captured from Android mobile devices. On the other hand, the choice for the number of iterations resulted in a less straightforward tradeoff between inference time and generated image quality. Excluding the relationship between the difficulty of the given text prompt, the number of iterations, and generated image quality and only focusing on the effect of the number of iterations in isolation, choosing numbers at and below 5 resulted in unacceptable deterioration. Thus, in the evaluation stage, values including and above 10 were considered.

Later, monocular depth estimation using MiDaS is again carried out, this time on the generated RGB image, such that the generated content can be matched to 3D geometric information and be brought to the 3D realm in a computationally efficient, simple, and fast way. As this depth map is again inverted and relative, it is registered to the reference depth provided by the client. While the objects and geometry of the generated image may not and should not follow the original image exactly, the closest proxy that can be used for registration is the client-side depth, thus this map is used. Then in the following Section 3.2.2 about mesh generation, this depth map and the generated RGB image are used for the final 3D mesh generation.

Image Generation Tradeoffs The stage of image generation requires a number of decisions to be made, under the following constraints:

1. Aiming to provide an experience to the user that is as real-time as possible (short inference time) - this results in a tradeoff between inference speed and generated image quality that is determined by the choice of the number of iterations

2. Making sure that depth map estimate for the generated RGB image results in depth values that do not diverge significantly from the client-side reference depth map. If they do diverge, we prefer the generated depth to be smaller than the reference depth such that the resulting 3D environment would rather be "in front of" the real environment than "behind". The idea behind this is to evaluate the likeliness of accidents that may occur during the AR experience due to the misrepresentation of the distance to objects in the environment, where real objects that are closer may be visually blocked by virtual objects appearing to be further away.

3.2.2 Mesh Generation

Once the generated RGB image and its registered depth map are available, in combination with the camera intrinsics received from the client side, they are used to create a point cloud. This step is carried out using the Open3D Python library. Afterward, the functionalities and algorithms provided by the Python library version of the MeshLab [19] framework are used for further processing and reaching a final mesh. The following sequence of steps is given below:

1. To make further processing faster, the point cloud is then subsampled.

2. As the point cloud is generated from a single frame, it is bound to be of roughly rectangular shape with a clear back and front side, where the normals must point towards the front defined by the user's position. Normals of the point cloud are computed based on the camera viewing direction such that they would not point in the back direction or opposite of the direction in which the camera is placed.

3. For constructing a mesh from the point cloud, we apply the algebraic point set surface (APSS) [20] variant of the marching cubes algorithm[21]. This algorithm uses marching cubes to extract the iso-surface of a moving least-squares (MLS) surface defined by the given point cloud.

4. Before applying any further processing steps, we again aim to reduce the mesh size such that both server-side processing and client-side memory and rendering requirements are at acceptable levels. Therefore, we decimate the mesh using quadric edge collapse [22] constrained by not changing mesh topology to avoid introducing holes and not changing surface normals.

5. The initial normal estimates used for the marching cubes algorithm could be noisy, causing artifacts on the generated mesh. We aim to reduce these by applying Laplacian smoothing[23].

6. The originally devised pipeline also included a step for hole closing, however, due to issues related to its PyMeshlab implementation, the current pipeline does not include it.

7. As the marching cubes implementation of PyMeshlab loses vertex color information, at this stage, the vertex color attributes of the original point cloud are transferred to the current mesh. A texture map is parametrized based on this transferred color information.

8. Mesh normal values are renormalized to avoid any numerical issues that may occur on the client side mesh construction and rendering.

3.3 Client Side

The main purpose of the client app is to receive requests from the user through its UI, capture relevant data required for generating an AR experience, communicate with the Server to order the generation of said environment and request it from the server when it is ready and finally deploy AR environments created by server. The specification for the client app must be such that it is able to perform functionalities required for an AR Experience such as maintaining a robust representation of the surrounding scene and the device. Additionally, the type of application we are building requires dynamically generating an environment during runtime and that the environment should match the existing geometry of the real environment. Therefore, the AR system that the app uses must support algorithms to obtain geometric cues to be sent to the server where they can be used as a reference point for generation, e.g. sparse or dense, complete or incomplete depth maps. It must also have the necessary rendering capabilities to construct and render a new 3D environment during runtime based on received files.

The most common and widely preferred SDK available for AR experience development on Android Devices is Google's ARCore SDK. ARCore incorporates motion tracking, environmental understanding, and light estimation into its system. It primarily relies on the detection of feature points along the device movement to locate the device, in an approach similar to SLAM. The feature points are then used to detect planes, which are then used to form a representation of the environment, in a similar manner to RANSAC [24].

In terms of a smooth and reliable experience for the user, an accurate and reliable representation of the environment and accurate motion tracking are key qualities. ARCore's visual-inertial odometry system is comparable if not better performing in some cases compared to similar alternatives such as Apple's ARKit, Intel RealSense T265 and Stereolabs ZED 2 according to a

benchmark study comparing these four systems [25]. The study gives a slight lead to ARCore over ARKit in indoor scenes, while ARKit takes the lead in outdoor scenes. Given that our initial prototype is primarily intended for indoor settings, these results present ARCore as a valid choice.

Among other functionalities, ARCore's anchor system for object placement and its Depth API's depth estimation system are tools that we needed for our implementation. ARCore introduced Depth API for improved immersion of experiences, allowing for the management of occlusion between the real environment and virtual objects. The Depth API forms its depth maps through a stereo matching approach that optimizes a combination of PatchMatch Stereo[26] and HashMatch[27] algorithms. Although the produced depth map is rather coarse and does not capture high-frequency details well, it works as a valid ground truth to be refined and used on the server side. It is also important to note the chosen algorithms may perform poorly depending on illumination conditions, reflection or scattering caused by material properties of the environment as well as lack of distinctive textures in the environment (e.g. large mono-color surfaces) [25].

Two platforms supporting ARCore SDK based AR applications on Android devices, namely Android Studio and ARCore, were considered for the client application implementation. App prototypes were implemented on both. After considering the capabilities of their rendering systems, availability of abstractions, documentation, active developer community, and potential to also support Apple devices in the future, Unity was chosen as the preferred development medium. Unity provides its own wrapper, Ar Foundation, around ARCore. This makes it more straightforward for supporting Apple devices as the same wrapper applies to ARKit as well, which is the corresponding AR development library for Apple.

4 Experiments & Results

Our system evaluation incorporates quantitative evaluation and a user study. The quantitative evaluation is carried out on the labeled subset provided in the NYU Depth Dataset V2, which is an indoor dataset consisting of 1449 samples of ground truth RGB images and depth maps captured by Microsoft Kinect as well as segmentation maps [28].

The stages of our quantitative evaluation are meant to capture the amount of error or distortion introduced at each step of the generation pipeline, determine which stages are the bottleneck in producing a robust final mesh, and inform implementation decisions.
Our user study aims to capture how users quantify their experience of the 2D generation section of the pipeline, as well as to determine the impact of implementation decisions on user perception of generation quality.

The implementation decisions we consider in both the quantitative evaluation and user study are primarily along two axes: the number of iterations of the generative model and the type of conditioning applied during generation. As explained in Section 3.2.1, these two axes can be mapped to visual quality vs. speed and visual quality vs. depth accuracy trade-offs. Having comparative evaluations along these two axes in the 2D evaluation allows us to inform our final system implementation accordingly, later supporting the quantitative evaluation with a user evaluation. Our quantitative evaluation methodology consists of depth, generated RGB image, point cloud and mesh evaluation. Our user evaluation consists of generated RGB image evaluation.

4.1 Depth Evaluation

As explained in Section 3.2, we aim to infer high-fidelity depth from the generated image such that it can be overlaid onto the surrounding environment in an immersive manner. Therefore, we aim to determine how depth values diverge between ground truth and monocular depth estimation based on the original RGB image, as well as monocular depth estimation on the generated RGB image. By analyzing this divergence for different versions of our pipeline we aim to understand how well the generated mesh will follow the structure of the real environment, and how implementation decisions will impact the divergence.

Overall, we carry out two depth comparison experiments, the first one given in Table 1 determines the effect of the number of inference steps on the inferred depth maps, while the second experiment given in Table 2 determines how different condition types used for ControlNet effect the inferred depth maps

We carry out a comparative depth evaluation with three pairs of depth maps for each comparison experiment. We begin by comparing the ground truth (GT) depth map to the predicted ground truth depth map (PGT) (GT-PGT), where we use "ground truth" to refer to the fact that the depth map corresponds to the input RGB image and not the generated RGB image. The PGT depth map is the inference result of the monocular depth estimation model of the pipeline, which is MiDas. This step aims to capture how much error is introduced when obtaining the depth map (PGT) that will be used as the conditioning image for the ControlNet model. Then, after the generation of the RGB image, we carry out our PGT vs generated depth map (G) (PGT-G) evaluation. This step aims to capture how much the depth map of the generated image deviates from the conditioning image used for generation. Finally, we have a GT-G comparison, where we see the final error between the initial ground truth and the final depth map used for creating the end result. We expect these three comparisons to yield increasing error.

For evaluating and comparing the depth maps, among the standard metrics used for benchmarking monocular depth estimation models [29] we use Root Mean Square Error (RMSE) and accuracy with a threshold (δ) as well as introducing a new metric for the specific purpose of the project, which can be referred to as the average behind error (Behind-Avg). While we record all other evaluation metrics, the given metrics were found to be the most informative for the purpose of a comparative evaluation with the end goal of being used in an AR system. As we can present RMSE in terms of meters, it provides an error value that has a meaningful scale and can be thought of in the context of an indoor AR environment. Meanwhile, the accuracy values can provide a normalized way of comparing the results of different design choices. Finally, the average behind error is defined as the average distance between the pixels of the depth map and the reference depth map only for the pixels that have higher depth values than the corresponding reference pixels. In other words, only the sections of the depth map that are "behind" the ground truth depth map are taken and their average distance to the corresponding ground truth pixels is retrieved. The aim of using such a metric is to have a numerical value representing how "behind" a generated depth map is from the ground truth, since in an augmented reality or virtual reality environment, virtual objects that are further away than the real environment but visually mask the real environment may cause a user to collide with real objects. Thus, it is a metric introduced for the purposes of user safety and a smooth AR experience.

4.1.1 Effect of the Number of Iterations on Depth Maps

Inspecting the outcome given in Table 1, we can clearly see that running ControlNet inference for a larger number of iterations does not reduce the depth error introduced at the RGB image generation stage. Neither of the metrics improves from the results for 10 iterations upwards, either staying the same or dropping. However, even if there was a visible improvement in the metrics as the number of iterations increases, the standard deviations for thresholded accuracy and average behind error are too high for declaring it a strong enough reason to prefer a higher number of iterations. It is also important to note that across all numbers of iterations and for the overall table, the GT-PGT section has worse results than PGT-G except for RMSE values from the iteration numbers 20 and 30. Recalling that this section represents the discrepancy introduced at the initial monocular depth estimation on the ground truth RGB image followed by registration to the ground truth, we can conclude that the bottleneck in depth quality and possibly any following issues relating to mesh quality and AR experience safety is introduced at this early stage. Then, assuming that all other parameters of ControlNet are kept constant and the number of iterations is set to 10, the highest improvement can be achieved by improving the monocular depth model itself or the registration process through a different registration process.

4.1.2 Effect of the Condition Type on Depth Maps

Looking at the results in Table 1, the first observation to be made is that pure depth conditioning results in the best depth quality of all conditioning approaches, which is to be expected. On the other hand, we would expect the combination of depth and segmentation conditioning to perform better than segmentation conditioning and worse than depth conditioning. This is the case for the RMSE metric while being closer to that of the segmentation result than that of the depth result. However, for accuracy and average behind error, we see that the combined conditioning in fact results in a clear deterioration. This may be a side effect of the ControlNet model struggling overall with image generation when under multiple constraints with the specific form of conditioning applied by the ControlNet pipeline. This may not mean that e.g. if a new Stable Diffusion model were to be trained from scratch to use the combination of both conditions it would perform worse than another model trained with pure depth. If the additive nature of muli-condition ControlNet conditioning is the culprit, we are likely to see a similar picture in 2D RGB image evaluation results.

4.2 2D and 3D Evaluation

Our generated RGB image evaluation aims to capture image quality in terms of structure and diversity as well as the ability to adhere to the text prompt used for generation. For quantifying adherence to the text prompt we use the standard metric used for Text2Image models which is CLIP Score[30], a metric obtained from the cross-modal model CLIP trained on image-caption pairs. Notably, Text2Room[14] also uses this metric, albeit on data of different scales produced through a different process. For measuring the quality of generated images, Inception Score (IS) is used in the Stable Diffusion [6] and Text2Room [14] papers. The metric is derived from the output of a pretrained Inceptionv3 image classification model and evaluates how well each given sample image fits the distribution of the images that the model has seen before [31]. Whether images that we intend to use for our purposes should fit well into the training distribution of IS is in fact not clear since we may explicitly want images that look out of distribution compared to an ordinary image dataset when we create an arbitrary AR experience. However, IS may still

be useful in comparing the perceptual quality of different pipeline variants, while perhaps not being useful in this case to compare with other papers. While another similar metric, FID, has gained traction for comparing image distributions [31], for the reason that we do not aim to fit an input distribution, we do not use this metric. The final metric used for 2D RGB evaluation is LPIPS, which uses the features of again a pre-trained model to estimate the diversity of generated samples[31].

For the evaluation of the 3D point cloud and mesh generation, we look into how the point cloud of the generated image diverges structurally from the ground truth point cloud and how the final mesh of the generated image diverges from the point cloud of the same image. We quantify this divergence using Hausdorff distance [32], a distance metric used between sets of points that is also used for evaluating shape completion fidelity in the SDFusion paper[10].

4.2.1 2D RGB Image and 3D Results

The results for this section are found in Table 3 When it comes to adherence to the text prompt, we can clearly see that the segmentation conditioning gave the best results, which will later be confirmed through the user evaluation results in Section 4.4 as well. As explained there as well, we may attribute this to segmentation conditioning being a less restrictive form of conditioning that does not specify the exact geometric structure of the scene but rather gives outlines of semantic regions. The optimization required to interpret these loose outlines may be an easier task than fitting to a dense depth map. While the inception score results suggest a lead for depth conditioning, it must be considered that standard deviation is high compared to the score differences between different condition types. While for depth and segmentation conditioning variants, a tendency to plateau or improve in the metrics is more prevalent as the number of iterations increases, for the combined conditioning variant, a deterioration in quality is observed.

Looking at the Hausdorff distance results, we see that depth conditioning gives the best fidelity to the geometry of the point cloud derived from the original RGB image. This was expected given the similar lead that depth conditioning had in the comparative depth evaluation in Table 2. It is also clear that the vast majority of divergence between the original point cloud and the final mesh happens due to the divergence of the point cloud derived from the generated image and not due to the processing steps between this point cloud and the final mesh.

Considering all metrics together and their uncertainties, we conclude that this quantitative evaluation may not be sufficient to give an absolute lead to a conditioning type in terms of structural quality and visual appeal, and considering the results together with the user evaluation stage will be more beneficial. In terms of geometric fidelity to the ground truth, however, depth conditioning is the leader. While we expected the combined conditioning to give better Hausdorff distance results than only segmentation conditioning, this was not the case. As the combined conditioning become difficult to operate due to GPU memory limitations, runtime measurements could not be made using the final server we deployed. As the results for this conditioning option are not competitive in any case, this variant was not included in the user study that followed the benchmark evalution.

			Number of Inference Steps					
			10		20		30	
Eval Metric		GT-PGT	PGT-G	GT-G	PGT-G	GT-G	PGT-G	GT-G
RMSE (m) ↓		0.474 ± 0.5	**0.433 ± 0.5**	**0.559 ± 0.55**	0.5 ± 0.725	0.599 ± 0.625	0.534 ± 0.85	0.655 ± 0.767
Accuracy $< \delta \uparrow$	1.25[%]	89.4 ± 12	**91.2 ± 12**	**83.4 ± 11**	91.1 ± 11	83.2 ± 14	90.7 ± 12	82.9 ± 15
	1.25^2[%]	98.0 ± 4.6	**98.7 ± 4.6**	**96.6 ± 3.6**	98.5 ± 3.8	96.5 ± 5.8	98.6 ± 3.8	96.5 ± 6
	1.25^3[%]	99.5 ± 1.7	**99.7 ± 1.7**	**99.1 ± 1.4**	99.6 ± 1.6	99 ± 2.5	99.6 ± 1.6	99 ± 2.5
Behind-Avg (m) ↓		0.147 ± 0.15	**0.131 ± 0.15**	**0.169 ± 0.14**	0.151 ± 0.56	0.189 ± 0.58	0.138 ± 0.18	0.174 ± 0.2

Table 1: Comparative depth evaluation on the NYU dataset for the depth conditioned pipeline. Evaluation is carried out for different number of inference steps for the generative model. Three depth map types are used, GT (Ground Truth) referring to the reference LIDAR depth obtained from the dataset, PGT (Predicted Ground Truth) referring to depth maps obtained by monocular depth estimation on the original RGB image, G (Generated) depth map referring to the depth map estimated by monocular depth estimation on the RGB image generated by the generative model. GT-PGT captures the divergence introduced at the initial depth estimation stage, PGT-G captures the additional depth divergence introduced by image generation, and GT-G captures the end-to-end divergence between ground truth and final generated depth

			Conditioning Types for ControlNet					
			Depth		Segmentation		Depth + Segmentation	
Eval Metric		GT-PGT	PGT-Gen	GT-Gen	PGT-Gen	GT-Gen	PGT-Gen	GT-Gen
RMSE (m) ↓		0.474 ± 0.5	**0.433 ± 0.5**	**0.559 ± 0.55**	1.04 ± 9.73	1.09 ± 9.71	0.841 ± 1.37	0.855 ± 1.24
Accuracy $< \delta \uparrow$	1.25[%]	89.4 ± 12	**91.2 ± 12**	**83.4 ± 11**	81.5 ± 17	75.4 ± 18	67.1 ± 23	62.6 ± 22
	1.25^2[%]	98.0 ± 4.6	**98.7 ± 4.6**	**96.6 ± 3.6**	96.1 ± 7	94 ± 9	89.7 ± 14	87.5 ± 14
	1.25^3[%]	99.5 ± 1.7	**99.7 ± 1.7**	**99.1 ± 1.4**	99.1 ± 3	98.4 ± 4	97 ± 6	96.1 ± 7
Behind-Avg (m) ↓		0.147 ± 0.15	**0.131 ± 0.15**	**0.169 ± 0.14**	0.165 ± 0.3	0.182 ± 0.3	0.20 ± 0.19	0.208 ± 0.17

Table 2: Comparative depth evaluation of different condition types applied to the generative model. The depth map pairs evaluated are the same as given in 1. The condition types evaluated are depth map, segmentation map, and a multi-conditioning version where these two maps are combined with a higher weight given to the depth map.

		Depth Cond. Num. of Inference Steps			Segmentation Cond. Num. of Inference Steps			Depth + Segmentation Cond. Num. of Inference Steps		
Eval Metric		10	20	30	10	20	30	10	20	30
2D Evaluation	CLIP ↑ [0, 100]	23.78 ± 3.6	24 ± 3.7	23.98 ± 3.68	24.3 ± 4.36	**25.1 ± 3.99**	25.05 ± 3.86	23.45 ± 3.53	23.72 ± 3.53	24.03 ± 3.44
	LPIPS ↓	**0.697 ± 0.05**	0.7 ± 0.057	0.7 ± 0.058	0.715 ± 0.05	0.72 ± 0.049	0.72 ± 0.049	0.718 ± 0.046	0.718 ± 0.046	0.722 ± 0.047
	IS ↑	6.09 ± 0.36	6.167 ± 0.39	**6.46 ± 0.4**	5.849 ± 0.44	6.11 ± 0.47	6.2 ± 0.64	6.189 ± 0.51	5.98 ± 0.57	5.74 ± 0.82
3D Evaluation	PCD Hausdorff Dist. ↓	**0.13 ± 0.1**	0.133 ± 0.097	0.134 ± 0.099	0.164 ± 0.1	0.186 ± 0.132	0.189 ± 0.12	0.28 ± 0.2	0.283 ± 0.196	0.27 ± 0.182
	Mesh Hausdorff Dist. ↓	0.02 ± 0.13	0.019 ± 0.135	0.018 ± 0.12	0.062 ± 0.22	**0.0136 ± 0.063**	0.0195 ± 0.076	0.022 ± 0.126	0.0198 ± 0.098	0.025 ± 0.135
Total Runtime (seconds) ↓		69 ± 12	119 ± 40	136 ± 29	91 ± 26	133 ± 41	142 ± 31	N/A	N/A	N/A

Table 3: Comparative 2D RGB Image and 3D point cloud and mesh evaluation table. PCD Hausdorff distance represents the Hausdorff distance between the point cloud extracted from the original RGB + PGT depth and the point cloud extracted from generated RGB + G Depth. We aim to capture the divergence introduced by image generation with this distance. The mesh Hausdorff distance represents the distance between the final mesh and generated point cloud. We aim to capture how much divergence the marching cubes algorithm and the following mesh processing steps cause from the generated point cloud to the final mesh.

4.3 2D User Evaluation

The 2D user evaluation is intended to capture how well the quantitative evaluation made using the metrics used in the previous sections measured up to the perception and preferences of real users. The evaluation was carried out as a Google Form. Participants were asked to answer every question before submitting and were allowed to select multiple options in questions that were comparative in nature. All questions were 2D RGB image based. In comparative questions, the labels of the images were not provided to participants. Both the order of images within comparative questions and the order of the questions within each section were randomized. The evaluation was completed by 20 adults that were expected to have a graduate-level academic background. Question samples can be found in Figure 2 The evaluation posed questions that explore:

1. Which number of inference steps for the depth-conditioned Controlnet model result in generated images that are preferred by the user. Given in Table 3a

2. Keeping the number of iterations fixed at 10, the Controlnet variant with which condition type among depth and segmentation maps results in generated images that are preferred by the user. Given in Table 3b

3. Keeping the number of iterations fixed at 10, the Controlnet variant with which condition type among depth and segmentation maps results in generated images that are preferred by the user. Given in Table 3b

4. Evaluation of the structural similarity of the generated image to the original RGB image on the Likert scale. Asked for both depth and segmentation conditioned generations applied to the same original RGB image. Given in Table 3c.

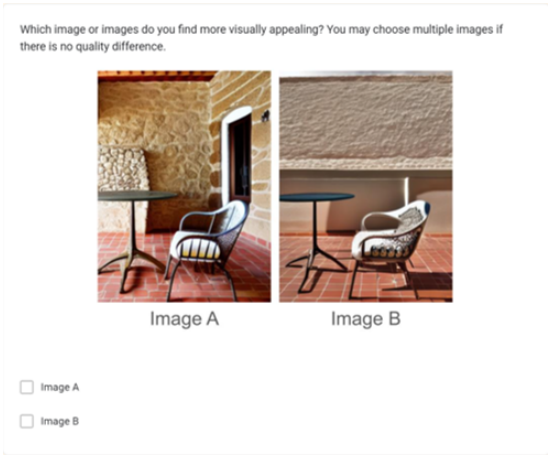
5. Evaluation of the adherence of the generated image to the given text prompt on the Likert scale. Asked for both depth and segmentation conditioned generations applied to the same original RGB image. Given in Table 3c.

4.4 2D User Evaluation Results

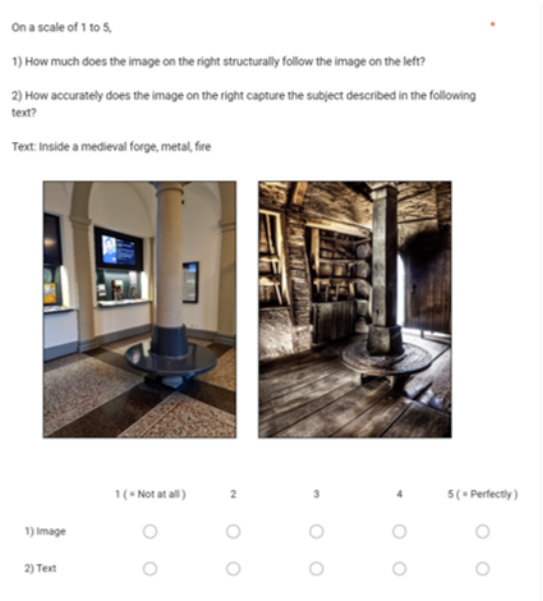
The results are presented in the form of box plots in Figure 3, where the responses are averaged per input image before conversion to a box plot. Thus, the data points represented in the plots show how each variable changes across images. An increase in the mean preference is seen with the number of inference steps in Figure 3a, however, the variance of the results is high enough that this may not be significant enough considering the added computational requirements of increasing the number of iterations. The lower error bar for 30 iterations goes as low as 10 iterations, showing that uncertainty about image quality does not reduce in this experiment as the number of iterations increases.

A visible preference appears for segmentation conditioned images in the comparative evaluation plot given in Figure 3b despite extremely high uncertainty. This result is in contradiction to the visibly higher structural similarity rating of depth conditioning compared to segmentation conditioning given in Figure 3c. Looking at this outcome from another perspective, we may consider that the segmentation condition constrains the structural composition of the image less as it only provides outlines of semantic regions rather than all geometric information. While this degrades structural adherence to the original image, it may give the model a less constrained

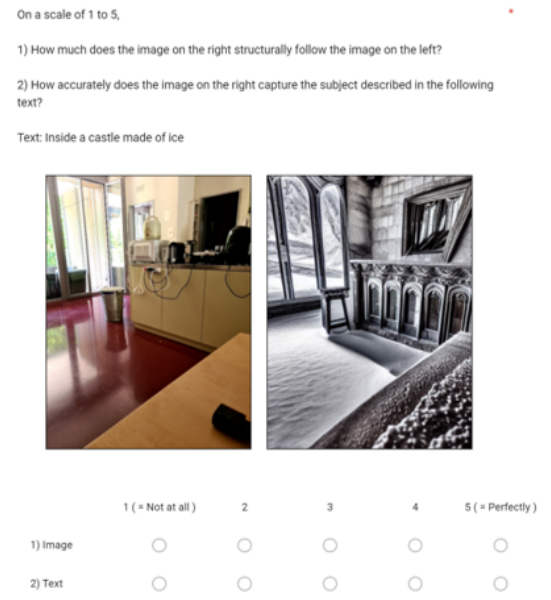
(a) This question aims to capture user preference of images generated with different numbers of iterations for ControlNet. In the sample question given, the mapping between images and the number of iterations is as follows: (A: 10, B: 30, C: 20)



(b) This question aims to capture user preference for images generated with different ControlNet variants using depth or segmentation map conditioning. In the sample question given, the mapping between images and the condition types is as follows: (A: segmentation, B: depth)



(c) This question carries out a Likert scale evaluation of structural similarity and text-prompt similarity for depth-conditioned image generation.

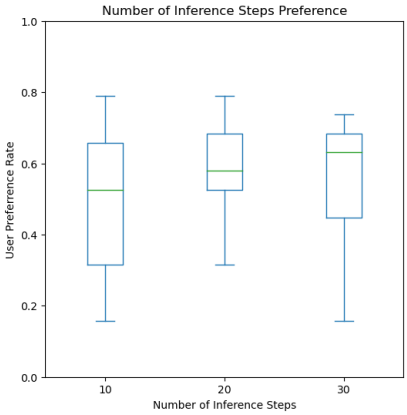


(d) This question carries out Likert scale evaluation of structural similarity and text-prompt similarity for segmentation conditioned image generation.

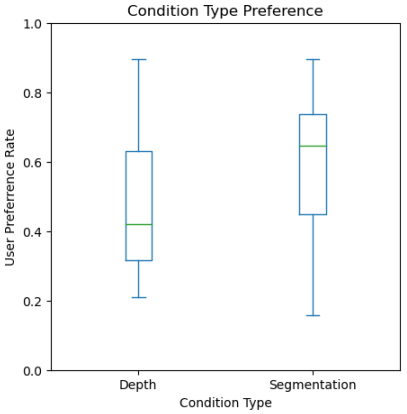
Figure 2 Sample questions taken from the evaluation form given to participants.

space to improve its generated image more easily. However, high variance is again observed for the segmentation case, suggesting that the detail and exact content of the segmentation map results in more varied outcomes according to users.

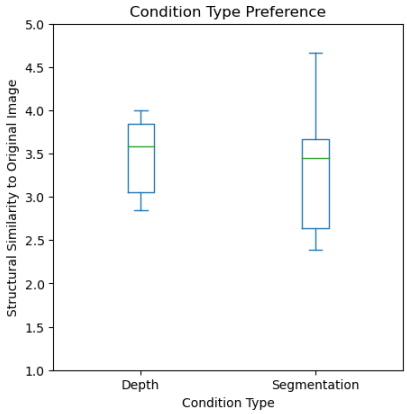
Finally, adherence to the text prompt, as visualized in Figure 3d gives a slight lead to segmentation conditioning. This may again be related to the possibility that segmentation maps constrain the generation space less strictly, allowing for the text conditioning to be applied more freely as well. If the exact shape and details of an object are known as is often the case with depth conditioning, it may be a more difficult optimization task to interpret this shape according to the text prompt.



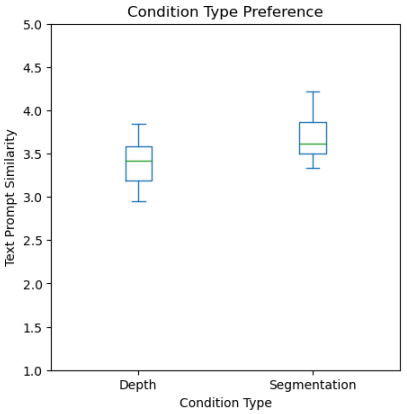
(a) User preference rate of images generated using different numbers of iterations for the generative model. The box plot was produced using the preference rate data averaged per individual image in the questionnaire.



(b) User preference rate of images generated using different condition types for the generative model. The box plot was produced using the preference rate data averaged per individual image in the questionnaire.



(c) User evaluation of the structural similarity of generated images to original RGB images on the Likert scale. The question was posed for images generated using both depth and segmentation conditioned ControlNet variants applied to the same original RGB images. The mean score for each image in the evaluation was taken and used for producing the box plot access image samples.



(d) User evaluation of adherence of generated images to the text prompt of on the Likert scale. The question was posed for images generated using both depth and segmentation conditioned ControlNet variants applied to the same original RGB images. The mean score for each image in the evaluation was taken and used for producing the box plot access image samples.

Figure 3 2D user evaluation results. Tables (a) and (b) are for sections where users were presented randomly ordered images with no information regarding their label (e.g. condition type or number of iterations) and were asked to declare a preference based on visual appeal, given the option to choose multiple images as well. Tables (c) and (d) are for sections of the 2D user study where participants were asked to quantify the following values on the Likert scale: Structural Similarity (SS) of the generated image to the original image, the Text Prompt Similarity (TS) of the generated image to the given text prompt.

4.5 Final Client Application Recordings

To observe how the generated mesh behaves after being rendered on the client application please refer to the videos found at this link. Otherwise, screenshots from the respective recordings are also available in Appendix A.2.

5 Discussion & Conclusion

5.1 Discussion of Evaluation Results

Given the advantage of pure depth conditioning regarding better preservation of geometric information, until the methods used for combining segmentation and depth conditioning advance in the following months, pure depth conditioning will be preferred for the purposes of our system. There is evidence coming from the user study to suggest that higher visual appeal is achieved with the segmentation conditioning when the original RGB image and the text prompt are kept the same. Given the importance of text prompt engineering to generation quality, future experiments exploring specific automatic prompt engineering techniques such as the semantic prompts for sections of a room used in [14] for these conditioning types may change this advantage. For example, a step of semantic segmentation to determine relevant objects in a scene and using this information to add further sections to the text prompt may be another way of introducing semantic information to a depth conditioned ControlNet based pipeline.

When it comes to the number of iterations used for ControlNet, while an improvement in 2d image quality is observed in the user study, the mesh generation stage of the pipeline loses more texture information than the marginal quality improvement gained with a higher number of steps. Thus, we use 10 as the number of iterations in the final pipeline. If future work improves on the mesh generation section of this pipeline, it may become more meaningful to use higher numbers of iterations given that the hardware used for the server keeps user waiting time at acceptable levels of e.g 30 seconds. The acceptable waiting time may be a part of an end-to-end evaluation of the final system if such an evaluation is to be carried out as part of future work.

5.2 Issues and Shortcomings

The main issue faced during the development of this project was related to the development environments of Android Studio and Unity which were each considered for the client application. The development and trial and error process with the rendering capabilities of Android Studio and the following change of development platforms took up a disproportional amount of allocated time. Issues that would need further investigation in this pipeline include improving the robustness of mesh generation, and exploring how to modify ControlNet parameters to enable better generation with the combination of depth and segmentation maps.

There are additional limitations independent of the decisions made on pipeline design. The algorithms used by ARCore to infer the client-side depth, drifts, and inaccuracies caused by ARCore's object placement and motion-tracking system and the limit that Unity has on the

number of vertices a single mesh may contain are such limitations. The approach of using monocular depth on the server side additionally introduces the inaccuracy of the monocular depth model, MiDaS, as a factor in geometric divergence between the real environment and the generated mesh. More accurate monocular depth models may be considered or a more accurate client-side stereo-matching system could replace the monocular depth inference on the original RGB image.

5.3 Future Work

As the field of diffusion-based generative models is highly dynamic with new papers, pipelines, and approaches being introduced every month, simply by the virtue of deploying more capable models, better generative results are bound to be obtained. For example, the release of ControlNetv2 brings the necessity of measuring any improvements that can be made by using this improved pipeline.

One of the most crucial stages where the final AR experience quality is reduced is the simplification and texturing of the produced mesh. An approach that does not rely on vertex colors for generating texture maps would allow for having larger planar surfaces without losing detailed texture. As it is currently in the pipeline, every vertex is mapped to a single color and decimation of the mesh inevitably reduces the detail of the texture. If this is avoided and large flat surfaces with detailed texture are achieved, the limited budget of total renderable vertices imposed by Unity and mobile hardware capabilities would be better used on regions of meshes that require a more detailed representation.

The most important next step following from this pipeline would be to carry it into a multi-view approach, representing an entire room using multiple meshes placed at separate anchors in the Unity AR Foundation environment. At that stage, depth completion and ControlNet-based conditioned image inpainting would be the primary tools for environment creation once the initial seed image is generated. A different approach may also be followed beginning with a comprehensive video recording of the environment, from which the server can partition sections, generate regions and reconstruct the entire environment in the form of multiple meshes, sending them to the user all together.

Depending on how the resource requirements of the pipeline evolve, an approach where the entire pipeline or its generative section is deployed on a serverless platform may be preferable if available hardware is insufficient.

Upon the completion of the client app for deployment to users, a final user evaluation of the overall application experience including sending requests, the waiting time for generation, generation quality, and the client side rendering, positioning, lighting, occlusion of the AR environment would provide a complete end-to-end evaluation of the system.

References

[1] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023.

[2] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.

[3] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models, 2022.

[4] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[5] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022.

[6] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[7] Chenshuang Zhang, Chaoning Zhang, Mengchun Zhang, and In So Kweon. Text-to-image diffusion models in generative ai: A survey, 2023.

[8] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv*, 2022.

[9] Ajay Jain, Ben Mildenhall, Jonathan T. Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields, 2022.

[10] Yen-Chi Cheng, Hsin-Ying Lee, Sergey Tulyakov, Alexander Schwing, and Liangyan Gui. Sdfusion: Multimodal 3d shape completion, reconstruction, and generation, 2023.

[11] Nasir Mohammad Khalid, Tianhao Xie, Eugene Belilovsky, and Tiberiu Popa. CLIP-mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia 2022 Conference Papers*. ACM, nov 2022.

[12] Zhengzhe Liu, Peng Dai, Ruihui Li, Xiaojuan Qi, and Chi-Wing Fu. Iss: Image as stepping stone for text-guided 3d shape generation, 2023.

[13] Rafail Fridman, Amit Abecasis, Yoni Kasten, and Tali Dekel. Scenescape: Text-driven consistent scene generation. *arXiv preprint arXiv:2302.01133*, 2023.

[14] Lukas Höllein, Ang Cao, Andrew Owens, Justin Johnson, and Matthias Nießner. Text2room: Extracting textured 3d meshes from 2d text-to-image models, 2023.

[15] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer, 2020.

[16] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation, 2021.

[17] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[18] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba.

Scene parsing through ade20k dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5122–5130, 2017.

[19] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[20] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. 26(3), 2007.

[21] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery.

[22] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, page 209–216, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[23] Olga Sorkine. Laplacian Mesh Processing. In Yiorgos Chrysanthou and Marcus Magnor, editors, *Eurographics 2005 - State of the Art Reports*. The Eurographics Association, 2005.

[24] R. Haenel, Q. Semler, E. Semin, Pierre Grussenmeyer, and Emmanuel Alby. Integration of depth maps from arcore to process point clouds in real time on a smartphone. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B2-2022:201–208, 05 2022.

[25] Pyojin Kim, Jungha Kim, Minkyeong Song, Yeoeun Lee, Moonkyeong Jung, and Hyeong-Geun Kim. A benchmark comparison of four off-the-shelf proprietary visual–inertial odometry systems. *Sensors*, 22:9873, 12 2022.

[26] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Patchmatch stereo - stereo matching with slanted support windows. volume 11, pages 14.1–14.11, 01 2011.

[27] S. Fanello, Julien P. C. Valentin, Adarsh Kowdle, Christoph Rhemann, Vladimir Tankovich, Carlo Ciliberto, Philip L. Davidson, and Shahram Izadi. Low compute and fully parallel computer vision with hashmatch. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3894–3903, 2017.

[28] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[29] Deep learning for monocular depth estimation: A review. *Neurocomputing*, 438:14–33, 2021.

[30] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning, 2022.

[31] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[32] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: measuring error on simplified surfaces. In *Computer Graphics Forum*, volume 17, pages 167–174. Blackwell Publishers, 1998.

A Appendix

A.1 System Implementation Codebase

The codebase of the system implementation consists of two sections, stored in separate GitHub repositories. The server implementation can be found at this link. The Android Unity client application implementation can be found at this link this link.

A.2 Supplementary Client Application Screenshots

Figure 4 Sample screenshots taken from the client app after receiving and rendering generated meshes in the AR experience.

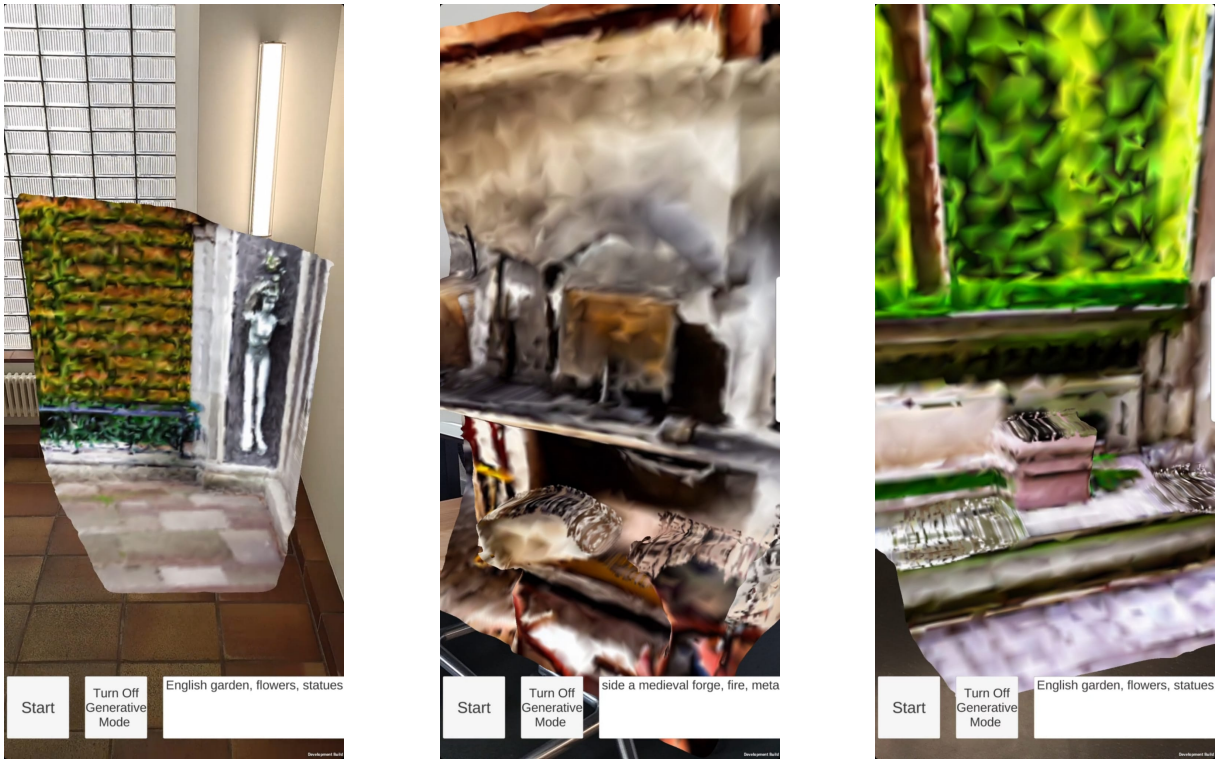


Figure 5 Sample screenshots taken from the client app after receiving and rendering generated meshes in the AR experience.