End of course project - Machine Learning for Computer Vision

Jacopo Meglioraldi, Eugenio De Luca
Master's Degree In Artificial Intelligence - University of Bologna

Abstract

With the growing popularity of online shopping, the ability to render 3D models quickly and efficiently has become increasingly critical for businesses. In order to grasp product and object main characteristics, usually just few images are needed. One of the most used techniques involve storing the relevant 3D models and using graphical engines to display them under different views. While this approach is computationally feasible, maintaining one or more 3D models for every product can become costly. In this project, we explore an alternative approach by generating a rendering image of a 3D model from a given point of view using Stable Diffusion [12] in conjunction with ControlNet [15], allowing us to condition the generation on an image of the desired 3D model and a specified rotation in three-dimensional space.

1 Overview

As outlined in the abstract, the objective of this project is to fine-tune Stable Diffusion, supported by ControlNet, to condition the generation process according to a staring view of a 3D object and the desired camera rotation in three-dimensional space. From ControlNet studies, it has been showed the ability to control the results of a diffusion model with additional information, in our case new views of the same object. To describe this process, we define the camera's position in spherical coordinates as $\rho = (r, \sigma, \phi)$. Additionally, let I_ρ^m be the image rendered from the 3D model m with the camera positioned at ρ. Given a rendering of the same object from a different camera position ρ' we define $\Phi = (r', \sigma', \phi')$ as the relative position such that $\rho + \Phi = \rho'$. Finally, let P be the prompt used for object m, the goal is to train a model G with parameters θ such that $G(I_\rho^m, \Phi, P, \theta) = I_{\rho'}^m$. In other words, the ControlNet model takes as input a specific point of view rendering image of a 3D model, a relative position Φ and a prompt P that describes the object and it outputs a new view of the same 3D model according to the new camera position $\Phi + \rho$.

2 Data

2.1 Dataset

The dataset used in this project is Objaverse-MIX [1], which contains 5,000 unique 3D models. In particular we used an already available dataset of images, where each objects was rendered in 12 images from fixed camera positions as shown in 1, resulting in a total of 60,000 512×512 images. Each image is paired with its corresponding camera extrinsic parameters, represented by 12 floating-point values: 9 for the rotation matrix and 3 for the translation vector.

In addition to the camera data, each image is associated with several metadata fields that provide further context of the object depicted. These fields include:

1. **Model ID**: A unique identifier string assigned to each of the 5,000 3D models.
2. **Name**: The name of the 3D model.
3. **Tags**: A list of descriptive tags, providing general information about the 3D model (e.g., material, software used to create it).
4. **Categories**: A list of categories associated with the model, such as "weapon," "animal," etc.
5. **Description**: A textual description associated with each objct. During our data exploration, we found that this field is often noisy: while some descriptions provide detailed information about the object, others contain incoherent text, multiple web-links or are left blank.

Figure 1: Example of the 12 renderings for a 3D model.

2.2 Camera rotation and position

Often to represent a camera position and orientation in space, a rotation matrix R and a translation vector T are required. Since the 3D object is the focus of our cameras, it is common practice to use an orientation which follow the OpenGL camera axis convention, called also "look-at" cameras. This particular orientation always *look at* the center of the scene, nominally the origin point O (0, 0, 0). Following this convention, cameras can be easily described just by their position in space T, because the orientation is univocal and can be computed using the following procedure:

- the $\hat{z}$ **axis** of the camera will be defined as the norm of the vector from the origin O to the camera position: $\hat{z} = \frac{T-O}{||T-O||}$
- the $\hat{x}$ **axis** is computed by the vector product between the $\hat{z}$ axis and the convention's $y*$ axis $(0, 1, 0)$: $\hat{x} = y^* \times \hat{z}$
- the $\hat{y}$ **axis** is finally computed as the vector product of the new found $\hat{x}$ and $\hat{z}$ axis: $\hat{y} = \hat{z} \times \hat{x}$

As suggested in [7] we used spherical coordinates to parameterize the position of the camera T, using the standard transformation $f : (x, y, z) \longmapsto (r, \sigma, \phi)$. Another advantage of this design choice is that translations in the spherical coordinate system are easier to understand for users with respect Cartesian coordinates.

2.3 View transform dataset creation

Once we converted the rotation matrix from Cartesian representation to spherical coordinates, we needed to create ground truth data for our model. At this point, for a given 3D model m, we have access to all the rendered images $I^m_{\rho_i}$, where ρ_i represents one of the 12 possible rotations for $i \in \{0, \ldots, 11\}$. As highlighted in the overview section, we now compute all possible rotations between pairs of rotations ρ_i and ρ_j for every pair (i, j) where $i \neq j$.

The rotation Φ between two points $\rho = (r, \sigma, \phi)$ and $\rho' = (r, \sigma', \phi')$ is defined by computing the difference between the corresponding spherical coordinates. Thus, we have:

$$\Phi = (r - r', \sigma - \sigma', \phi - \phi')$$

Working with angles allows us to bound the relative rotation between camera poses to be in the interval $[-\pi, \pi]$, by selecting the shortest path of translation between one position to another.

This approach allows us to compute 12×11 pairs for each 3D model, increasing our dataset to 660,000 data points.

2.4 Data reduction with CLIP

During data exploration, we noticed that some 3D models in the dataset were of insufficient quality for training due to poor rendering or lack of recognizability, as shown in Figure 2. Since the description fields were often incomplete or uninformative, we applied a filtering process to remove low-quality 3D models.

Figure 2: Example of a low quality 3D model.

We leveraged the capabilities of CLIP [10] to associate images with text by similarity score. Specifically, we computed a CLIP score (CS) for each 3D model m by comparing its renderings to the related text descriptions. The CLIP checkpoint used is hosted on HuggingFace's hub [8] that uses a ViT-B/32 architecture as its image encoder.

Given a 3D model m, we define its CLIP score with respect to a given text T as follows:

$$CS(m,T) = \frac{1}{12}\sum_{i=0}^{11} CLIP(I^m_{\rho_i}, T)$$

In other words, we averaged the CLIP scores of each of the 12 renderings of the object m with respect to text T. This averaging reduces noise and mitigates the effect of certain renderings being less recognizable due to their specific camera angles, as shown in Figure 3.

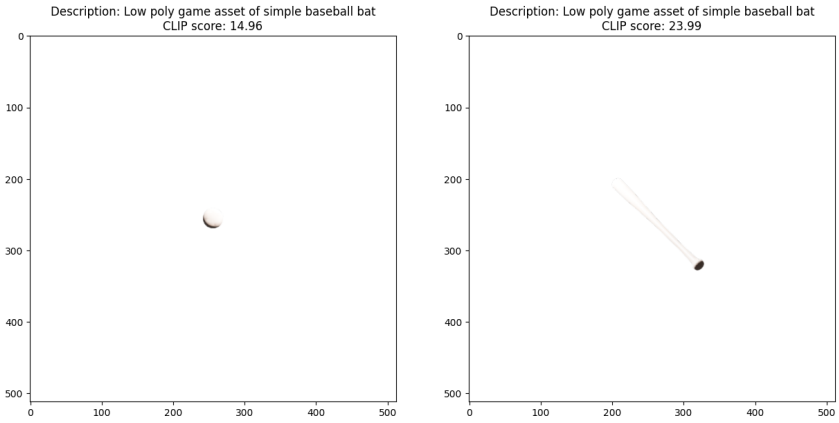

Figure 3: Example of the CLIP score discrepancies between two views of a good 3D model. In the first image, a white low poly baseball bat from a view perpendicular to its main axis. In the second image, the same white bat from a side view

For the text T, we analyzed six different options of combinations of text contained in the metadata of the objects. For each 3D model m we used as prompt the following options:

- **Description**: The description field of the 3D model.
- **Tags**: A comma-separated list of the 3D model's tags.
- **Categories**: A comma-separated list of the 3D model's categories.

Additionally, we concatenated pairs of these fields to form three combined texts options:

- **Description + Tags**
- **Description + Categories**
- **Tags + Categories**

Figure 4 shows that concatenating text fields did not lead to significant improvements in CLIP scores. Based on these findings, we chose to filter 3D models using the `Description + Tags` combination, removing any with a CLIP score below 24. This step resulted in the removal of 2,771 3D models from the dataset.

3 Models

3.1 ControlNet

Diffusion models have seen significant advancements, particularly in the ability to condition generation on inputs beyond simple text prompts. For instance, [14] introduced a method for conditioning generation based on sketches. ControlNet extends this concept by allowing conditioning on more complex inputs, such as images or specific features, while leveraging the knowledge embedded in pre-trained diffusion models.

In ControlNet, the core idea is to freeze the weights of a pre-trained Stable Diffusion model while training a network to handle the conditional inputs and compute new residuals for the different layers of the main model. This approach allows the original model to remain intact and keep the quality of the

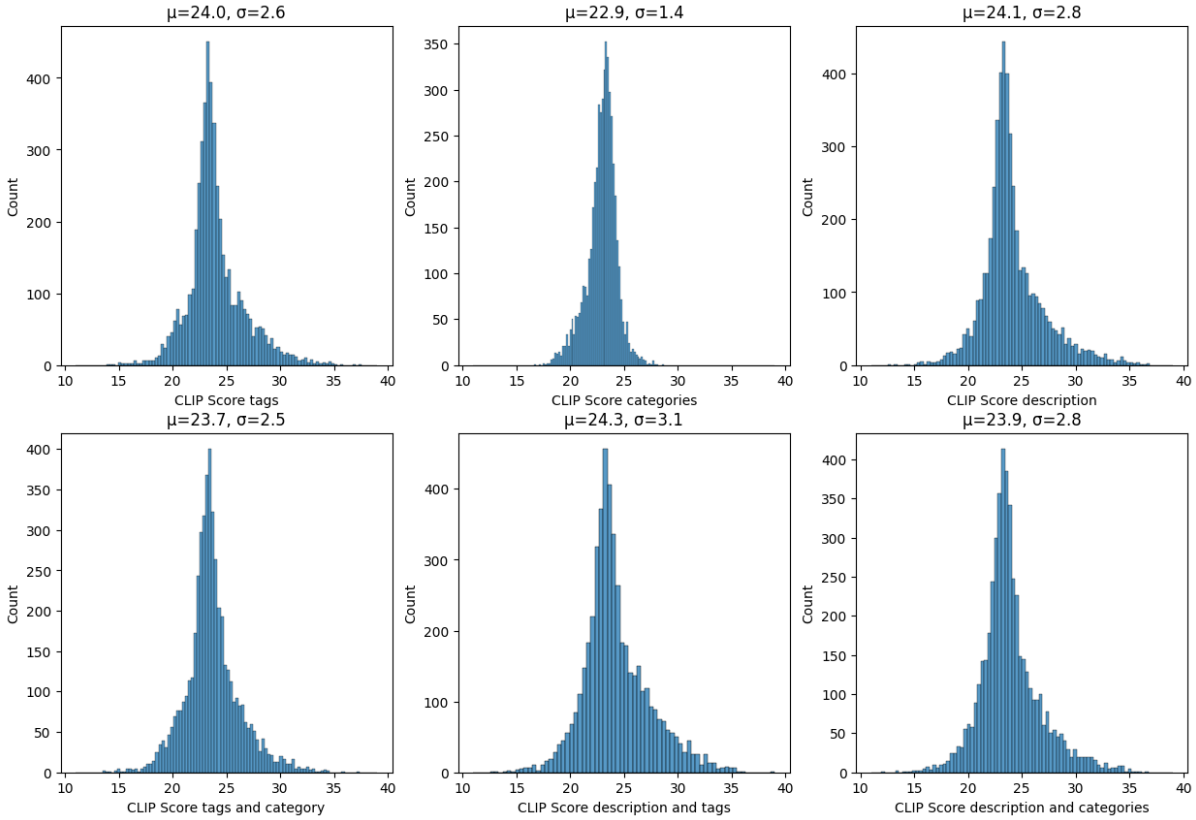

Figure 4: Average score for each 3D model according to given text.

generated images while the additional network guide the process to meet the conditions. Specifically, a copy of each neural block from the original network, with parameters denoted as Θ, is created and given new parameters Θ_c. These copies are connected to both the original blocks and the new conditional input through zero convolution layers, as shown in Figure 5.

Zero convolutions are 1×1 convolutions with weights and biases initialized to zero. This initialization ensures that, at the start of the training, ControlNet does not alter the output of the pre-trained model. In the first training step, the original model's output remains unchanged, as the zero convolutions output is zero. Over time, these convolutions are fine-tuned to allow the copied blocks to influence the original network without introducing unwanted noise.

Let ζ_1 and ζ_2 represent the zero convolution layers with parameters Θ_1 and Θ_2, respectively. The input to the model is x, and the conditioning input is c. The final output y_c of the conditioned model is computed as follows:

$$y_c = G(x, \Theta) + \zeta_2(G(x + \zeta_1(c, \Theta_1), \Theta_c), \Theta_2)$$

In this project, we chose ControlNet for its flexibility in handling conditioning inputs. The only requirement for the conditional input c is that it matches the shape of the original input tensor x. Given that 3D models in our dataset are based on real-world objects, we hypothesized that ControlNet could harness the pre-existing knowledge in the Stable Diffusion model to generate consistent views of these objects from different angles. This would allow us to hopefully produce high-quality 3D renderings from various perspectives with minimal additional computation and without training specific diffusion models to handle camera rotation conditioning, such as [13].

The Stable Diffusion checkpoint used is `stable-diffusion-v1.5` [11] hosted on the HuggingFace's hub that provides:

- The Stable Diffusion U-Net.
- The Text Encoder.
- The VAE.

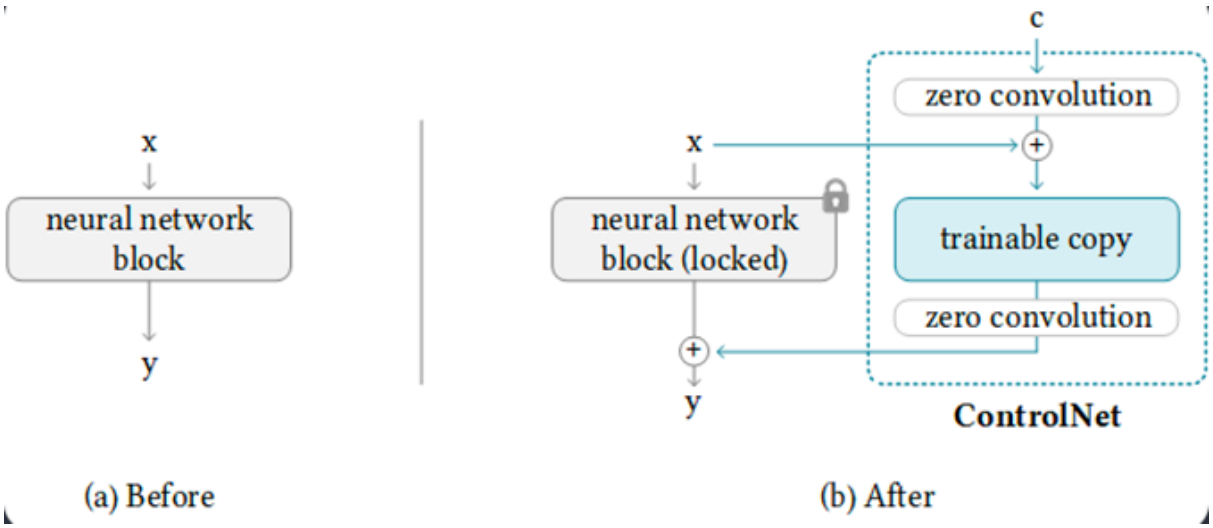

Figure 5: Structure of a ControlNet block. The copy is connected to the condition c with zero convolutions, and its output is fed back to the original model after processing through another zero convolution.

- The DDPM noise scheduler.

The ControlNet architecture is provided by HuggingFace's diffusers library [9].

3.2 Camera conditioning

To effectively utilize ControlNet for 3D rendering, we need to encode both the camera rotation Φ and the input image I_ρ^m into a tensor of the form $C \times H \times W$. We explored two different approaches to achieve this encoding.

3.2.1 Concatenation

The first approach creates the conditioning tensor c by concatenating the rotation Φ to the non-rotated image I_ρ^m. To ensure compatibility with ControlNet, it is necessary to convert $\Phi = (r, \sigma, \phi)$ from its original shape of 1×3 into a $C \times 512 \times 512$ format. This is accomplished using the `tile` function in Numpy (equivalent to `repeat` in PyTorch), which generates a $3 \times 512 \times 512$ tensor CC, where:

$$CC[i, :, :] = \Phi[i] \cdot \mathbb{1}_{512 \times 512}$$

Here, $\mathbb{1}_{512 \times 512}$ denotes a 512×512 matrix filled with ones, replicating each element of Φ across the spatial dimensions. The resulting tensor CC is then concatenated with I_ρ^m to form the final conditioning tensor c of shape $6 \times 512 \times 512$.

This method was chosen initially due to its computational efficiency and ease of implementation, requiring minimal transformation of the input data. One of the first step of the ControlNet architecture is a convolution network that will mimic the VAE image encoder of the stable diffusion pipeline in order to extract latent space feature from the conditioning image. By adding the camera values to the encoder, the extracted features will also take in consideration the camera pose.

3.2.2 Shift

The second approach exploit the idea that different views of the same object will have similar features. To condition the features of the conditioning image with the camera rotation, we applied a special instance normalization step to the output features of the camera conditioning encoder. In particular from the set of features F computed with shape $320 \times 64 \times 64$ we compute the mean μ and standard deviation σ of each channel. We then concatenate the resulting vectors with the camera relative position values, followed by a fully connected layer in order to return to the desired dimension. In doing so we compute μ' and σ' vectors which represent the mean and standard deviation of the features for the rotated image.

The normalization step shift the means and standard deviation of the features to match the desired ones

$$F' = \frac{F - \mu}{\sigma + \varepsilon}\sigma' + \mu'$$

This method was inspired by the Adaptive instance normalization layer [5] which successfully apply style transfer by a custom not learnable instance normalization layer.

4 Method

4.1 Splits

Train, Validation, and Test Splits To create our splits, we started by forming the test set. We selected four models from the dataset with the highest CLIP scores using the Description + Tags text fields (denoted as DT). After manual inspection, we chose models that were both visually interesting and simple in shape, as well as commonly seen in real-world images (see Figure 6).

Figure 6: Our test models.

The selected models (a chicken, a fire extinguisher, a doughnut, and a fire hydrant) were removed from the dataset for testing. For the training and validation sets, we aimed for an 80/20 ratio, adding models with the higher DT to the validation split until this ratio was achieved. For each model added to the validation set, we included all 132 renderings related to that specific model.

Sampling Despite using the CLIP strategy to filter the dataset, the remaining size was still too large for our available compute power. To address this, we sampled 20 renderings per model from the training set instead of retaining all 132. The sampling process ensured a uniform distribution of unique rotations to maintain their original one.

4.2 Training

The training loop The training loop employed to train ControlNet can be summarized as follows:

1. For each sample in the mini-batch, extract the conditioning image I^m_ρ, the ground-truth image $I^m_{\rho'}$, the conditioning camera rotation Φ, and the Description + Tags field.
2. Encode $I^m_{\rho'}$ using the VAE to obtain the latent representation x_0.
3. Sample a mini-batch of noise ϵ and select a random t between 0 and the total number of timesteps. Use the noise scheduler to compute ϵ_t, then add it to x_0 to obtain x_t.
4. Encode the Description + Tags text using the text encoder to derive the prompt P.
5. Generate the camera encodings according to the method being trained (i.e., either via concatenation or with an additional module) to obtain the condition c.
6. Run ControlNet using x_t, P, and c.
7. Run the Stable Diffusion U-Net using x_t, P, and the residuals computed by ControlNet to obtain the predicted noise $\hat{\epsilon}$.
8. Compute the loss as $MSE(\epsilon_t, \hat{\epsilon})$ and use it to update ControlNet's parameters.

This blueprint is an adaptation of `train_controlnet.py` from HuggingFace's diffusers library [9], as the original implementation was incompatible with our task.

Parameters The training was conducted with the following hyperparameters for both approaches:

- The optimizer used was `AdamW8bit` from the bitsandbytes foundation [2], a lightweight implementation of `AdamW` optimized for CUDA architectures.
- The batch size B was set to 4 with 4 gradient accumulation steps, resulting in an effective batch size of 16.
- The learning rate α was set to $B \cdot 5 \cdot 10^{-6}$.
- The `AdamW` weight decay was set to 10^{-2}.
- The `AdamW` epsilon parameter was set to 10^{-8}.
- A constant scheduler was used, with 500 warmup steps.

To improve training efficiency, we utilized HuggingFace's `Accelerate` package [6] to wrap ControlNet, the optimizer, scheduler, and dataloader, while Facebook Research's `XFormers` package [3] was used to enable memory-efficient attention computation.

Setup The concatenation model was trained for 3 epochs, while the shift model was trained for 2 epochs, with validation after each epoch. The training was performed on an intel i9-11900K octa-core Desktop Processor with Nvidia RTX 4090 24GB GPU and 60GB of RAM. Each epoch took 6-8 hours of training.

5 Results

5.1 Image generation

Our generation pipeline follows the standard Stable Diffusion blueprint:

1. The latents x_t are either taken from the previous step or randomly sampled if t is the first step.
2. The latent x_t is rescaled by the noise scheduler.
3. ControlNet is run with x_t and the condition c, thus obtaining the residuals R.
4. The Stable Diffusion U-Net is run on x_t and the residuals R, thus obtaining the noise ϵ_t.
5. If the guidance scale parameter [4] λ is greater than one, ϵ_t is updated as $\epsilon_t = \bar{\epsilon_t} + \lambda(\epsilon_t - \bar{\epsilon_t})$, where $\bar{\epsilon_t}$ represents the noise prediction at timestep t **without** the prompt conditioning.
6. ϵ_t is used by the noise scheduler to obtain x_{t-1}.
7. When this process is concluded (i.e., when $t = 0$), the latent x_0 is fed into the VAE to obtain the final predicted image.

5.2 Experiments

Effects of ControlNet The first experiment we did was actually testing how effective the ControlNet conditioning is. To do this, we predicted new images from our test set and compared them with predictions generated by the Stable Diffusion model without the ControlNet conditioning. The images were generated using 20 denoising steps and a guidance scale set to 7. Figure 7 shows the results of the shift model and figure 8 shows the results of the concatenate model.

The results clearly show that the ControlNet conditioning improves the results significantly. The model produces new images very similar to the input ones. It doesn't seem to reproduce the desired rotation, but it has indeed learnt to apply some rotation to the original point of view. From a qualitative perspective, the shift model predicts almost perfectly the doughnut sample while the concatenate model produces very promising result on the fire hydrant. Both of them struggle with the retention of the details: almost all the predictions present missing or inaccurate details, as well as different colors and shapes like in the case of the chicken model.

Figure 7: Results of the shift model. From left to right: conditioning image, predicted image with ControlNet, predicted image without ControlNet, target image.

Effects of the prompt In the next experiment we tried to perform the same type of predictions while completely omitting the textual prompt in the diffusion process to assert how it was steering the generation problem in the right direction. The predictions were performed using 20 denoising timesteps and with the the guidance scale set to 1 since we didn't need the model to closely follow any prompt. Figure 9 shows the results of the shift model and figure 10 shows the results of the concatenate model.

The results shows that the prompt in necessary for the model to have consistency between the input and the output since without it the generation seem to produce only unrecognizable objects. The Controlnet models seems to not have learned to extract features from the conditioning image in order to reconstruct the object alone.

Rotation tables For our next experiment we tried to create rotation tables; they are a grid of 5×5 images where every image is generated from the same input image I_ρ^m with different rotation Φ. To create these rotations we:

- Computed 5 angles α_i evenly spaced in the range $[-\pi, \pi]$.
- Computed 5 angles β_j evenly spaced in the range $[-\pi, \pi]$.
- Computed all the possible rotations $ROT_{i,j} = \Phi(r, \alpha_i, \beta_j)$ for every (i, j).
- Ran the model on every (i, j) starting from the same initial image.

All the predictions were performed with 20 denoising timesteps and the guidance scale parameter set to 7.

Figure 11 and 13 show the results of the shift model while figure 12 and 14 show the result of the concatenate model. Both models seem to express very slight rotations and both of them seem to struggle a lot with the chicken model: not only the rotation seem to apply only on the head in the case of the concatenate model, but they also seems to have an hard time in keeping the consistency with the original image. Regarding the fire extinguisher both model manage to keep the generation somewhat coherent with the original image while showing attempts to rotate the object in the various directions, with the shift model being the most solid on this regard. The rotations are clearly not perfect, but they indeed show some promise.

Figure 8: Results of the concatenate model.

CLIP Score Finally we used CLIP to evaluate the consistency of our models in retaining the original provided 3D models. To do this, for each sample s_i in the test set we:

- Predicted the target image with ControlNet and without it thus obtaining $I^m_{\rho'}$ and $\bar{I^m_{\rho'}}$.
- Processed these two image and the ground truth target image $\hat{I^m_{\rho'}}$ with the CLIP image encoder.
- Computed the predicted images' cosine similarity with the ground truth image.
- Averaged all the scores obtained this way.

We follow with the results:

	μ	σ
Stable Diffusion alone	0.9940	0.0035
SD + ControlNet - Shift	0.9965	0.0050
SD + ControlNet - Concatenate	0.9917	0.0072

Table 1: CLIP scores average and standard deviations for our two architecture with and without ControlNet.

Overall these results are very close and it's hard to draw insights from them. Surely enough, the models seems to keep consistency between the generated images regardless of the strategy used; surprisingly the presence of ControlNet doesn't really seem to increase the quality of the rendered models. This can be explained from the fact that CLIP considers two different chicken models similar regardless of some of their features like their color. Additionally notice that this scoring only rewards what model is rendered regardless of its rotation.

6 Conclusions

In this project we attempted to use stable diffusion guided by ControlNet to predict rotations in the 3D space of a given object while mantaining the output consistent. Sadly, while our results suggest that indeed some kind of rotational concept was learnt by ControlNet, this method didn't work well in the

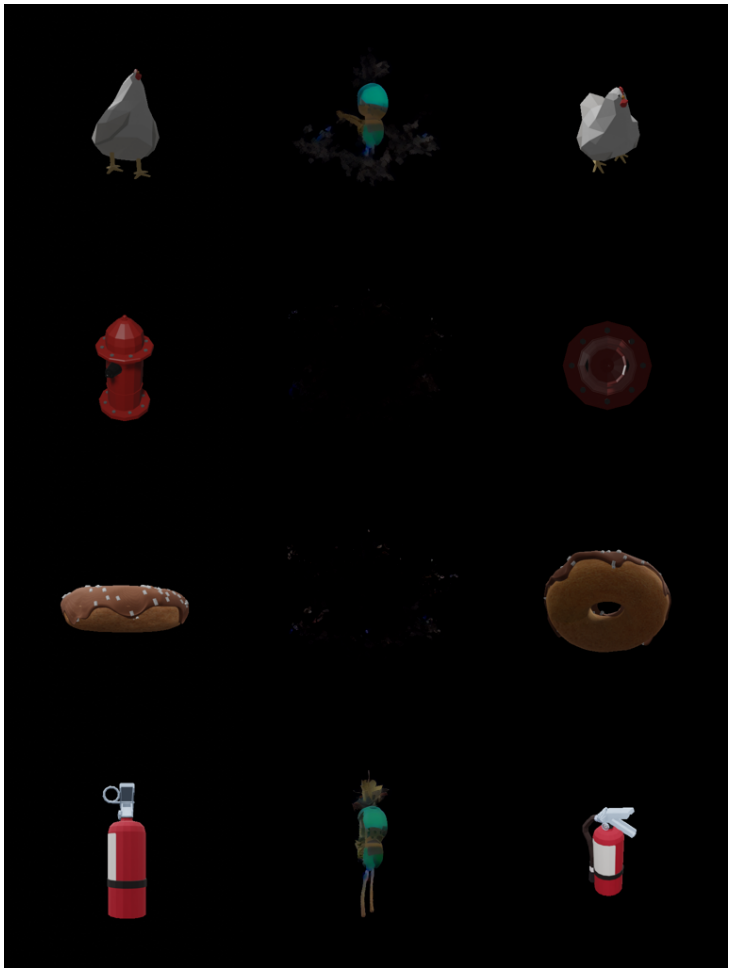

Figure 9: Results of the shift model without prompt. From left to right: conditioning image, predicted image with ControlNet, target image.

end for us. Our model struggled to keep inputs and outputs consistent and was only able to apply very small rotations to the given input objects without considering that most of the time the rotations were not the desired ones both in magnitude and direction. It is also true that due to the massive amount of compute needed for training we limited ourselves to train our two models for 2 and 3 epochs on heavily downsampled training data and this must have negatively affected our results. In the end we are still happy with the results we reached and we believe we should let ControlNet train for more epochs before calling this method off.

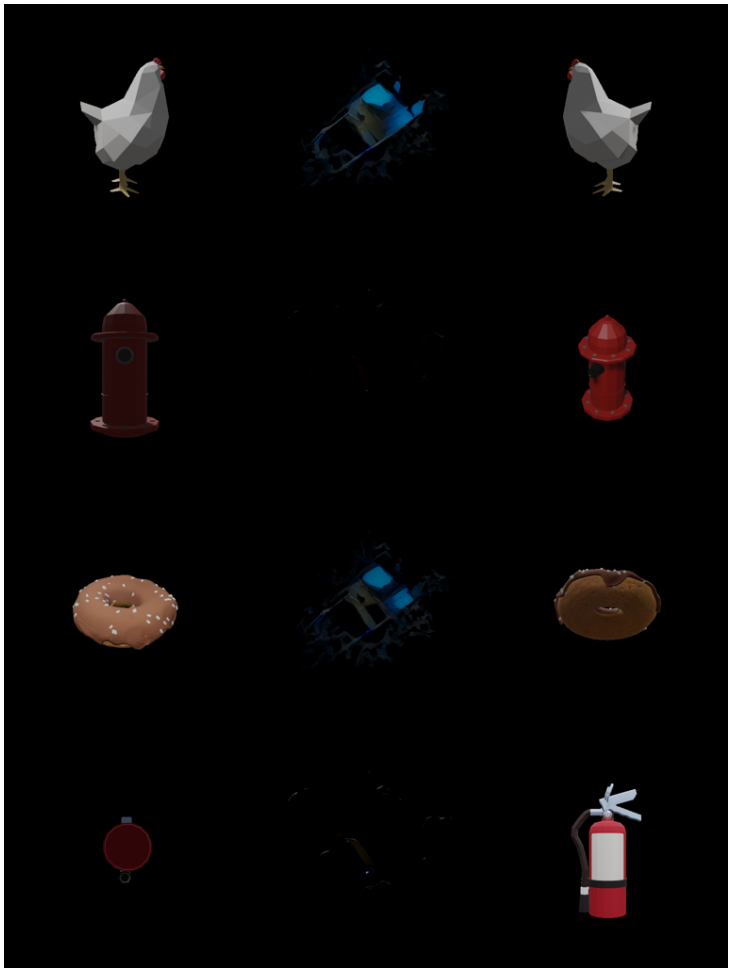

Figure 10: Results of the concatenate model without prompt.

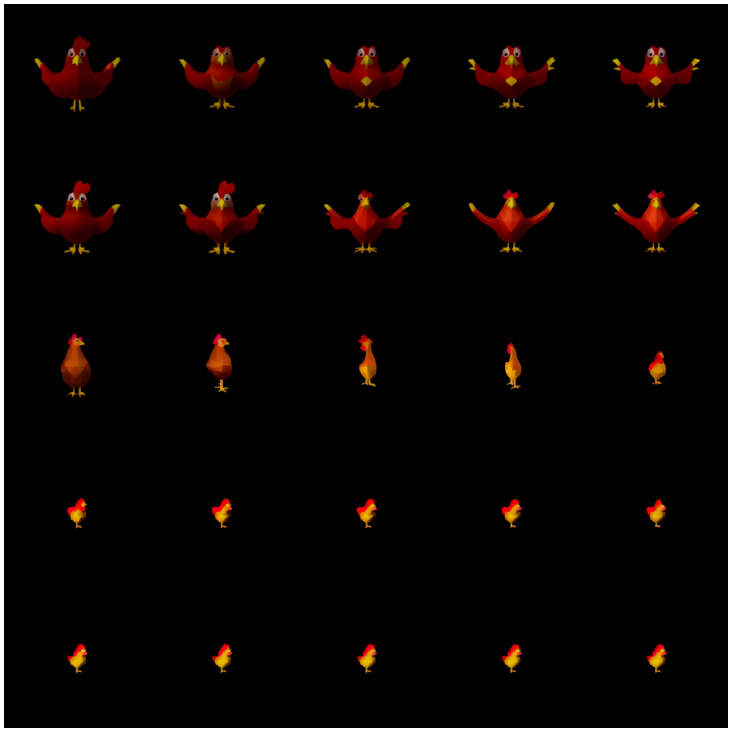

Figure 11: Rotation table of the shift architecture with the chicken model.

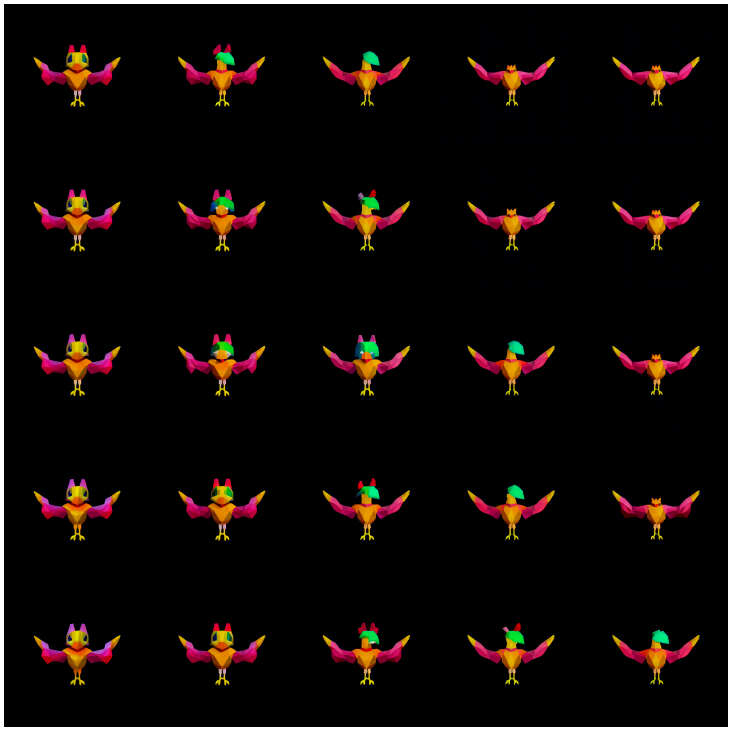

Figure 12: Rotation table of the concatenate architecture with the chicken model.

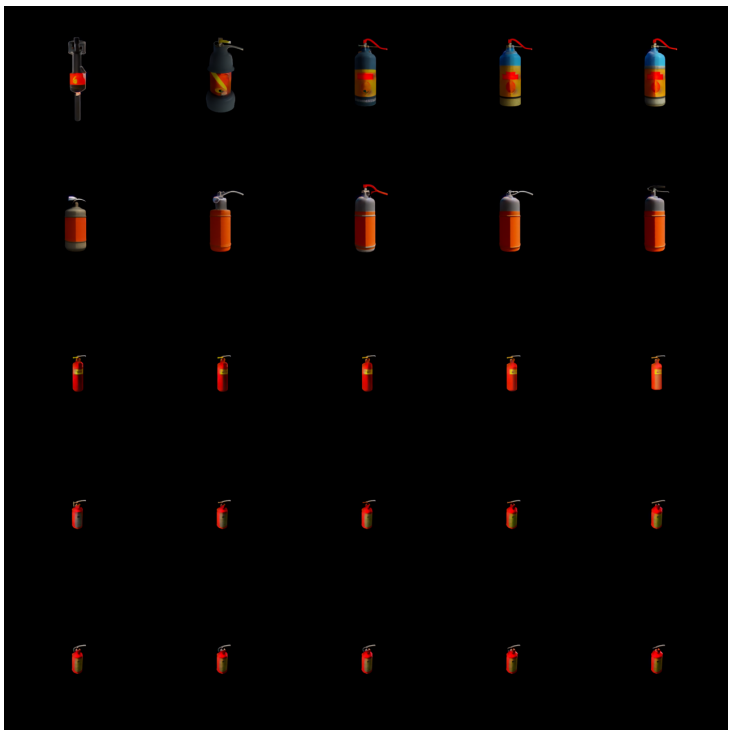

Figure 13: Rotation table of the shift architecture with the extinguisher model.

Figure 14: Rotation table of the concatenate architecture with the extinguisher model.

References

[1] Beijing Academy of Artificial Intelligence. *Objaverse-MIX*. https://huggingface.co/datasets/BAAI/Objaverse-MIX. 2023.

[2] bitsandbytes-foundation. *bitsandbytes*. https://github.com/bitsandbytes-foundation/bitsandbytes. 2021.

[3] facebookresearch. *xformers*. https://github.com/facebookresearch/xformers. 2021.

[4] Jonathan Ho and Tim Salimans. *Classifier-Free Diffusion Guidance*. 2022. arXiv: 2207.12598 [cs.LG]. URL: https://arxiv.org/abs/2207.12598.

[5] Xun Huang and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1501–1510.

[6] HuggingFace. *Accelerate*. https://github.com/huggingface/accelerate. 2020.

[7] Ruoshi Liu et al. "Zero-1-to-3: Zero-shot one image to 3d object". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2023, pp. 9298–9309.

[8] openai. *clip-vit-base-patch32*. https://huggingface.co/openai/clip-vit-base-patch32. 2021.

[9] Patrick von Platen et al. *Diffusers: State-of-the-art diffusion models*. Version 0.12.1. URL: https://github.com/huggingface/diffusers.

[10] Alec Radford et al. "Learning Transferable Visual Models From Natural Language Supervision". In: *CoRR* abs/2103.00020 (2021). arXiv: 2103.00020. URL: https://arxiv.org/abs/2103.00020.

[11] Robin Rombach et al. "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 10684–10695.

[12] Robin Rombach et al. "High-Resolution Image Synthesis with Latent Diffusion Models". In: *CoRR* abs/2112.10752 (2021). arXiv: 2112.10752. URL: https://arxiv.org/abs/2112.10752.

[13] Yichun Shi et al. "Mvdream: Multi-view diffusion for 3d generation". In: *arXiv preprint arXiv:2308.16512* (2023).

[14] Andrey Voynov, Kfir Aberman, and Daniel Cohen-Or. "Sketch-Guided Text-to-Image Diffusion Models". In: *ACM SIGGRAPH 2023 Conference Proceedings* (2022). URL: https://api.semanticscholar.org/CorpusID:254018130.

[15] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. "Adding Conditional Control to Text-to-Image Diffusion Models". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)* (2023), pp. 3813–3824. URL: https://api.semanticscholar.org/CorpusID:256827727.