Technology
Arts Sciences
TH Köln

Cologne University of Applied Sciences
Faculty of Computer Science and Engineering Science

M A S T E R T H E S I S

Texture Asset generation through Transformer Models

Cologne University of Applied Sciences
Campus Gummersbach
Master Digital Sciences

written by:
Dennis Gossler
11140150

First examiner: Prof. Dr. Olaf Mersmann
Second examiner: Prof. Dr. Boris Naujoks

Abstract

Blob

Contents

List of Figures

List of Tables

1 Introduction

Transformers are usally used to generate Text in like Large Language Model LLMs like the the GPT (Radford et al., 2019) / LaMma models (Touvron et al., 2023). This thesis will investigate the use of a traditional Transformer based architectures to generate texture assets. Traditional image generation is manly done throw stable diffision or throw Gan models To do so 3 different models are develop and trained each with a different twist to give the mane advantages of transformer based models to image texture generation. The dataset is also gathered to train the models. Thes sets of images are comming from many different sources. And needed to be cleaned and analyzed. These traind models should are capible to genreate basic floor texture that could be used in video games as assets. The develop models will be evaluated on a set of metrics and the results will be compared to other models. Additionally the development process will be shortly describes and the differentces to normal tipical programming workflow are illustrated.

1.1 Related work

1.2 Infrastructure for Model Development

To develop and train the models in this thesis, a powerful computing infrastructure is necessary to manage the extensive datasets and the substantial computational requirements for model training. Unlike conventional development environments where a standard laptop or desktop may suffice, most of the models in this thesis demand a more capable infrastructure. Therefore, a high-performance computing system situated in Berlin is used for the model training processes. This system contains an array of NVIDIA Tesla A100 80 GB GPUs, Ice Lake 8360Y CPUs and a significant quantity of RAM. Such a configuration, especially the substantial GPU memory, enables the training and execution of larger models that would be possible on a home workstation. The development of these models is carried out using Python and PyTorch, with the code being crafted in Visual Studio Code and managed through version control with Git. The model development and initial code testing are done on a local machine, reserving the high-performance system exclusively for the final training phases. This approach diverges from standard practices, where often both development and execution occur on the same development platform. Ensuring the code is free of errors prior to giving the task of training the model to the high-performance computing system is crucial, as discovering bugs in the training process can be exceedingly time-consuming. For instance, to endure a training session that extends for 30 hours, only to realize it terminated prematurely due to script errors.

1.3 Data

This section describes the methods used for gathering, cleaning, and analyzing data in a research thesis on textures. Essential for training a machine learning model, the data is carefully collected from various sources, cleaned to maintain uniformity, and examined for patterns, with a focus on color distribution.

1.3.1 Data Retrieval

On the internet, a wide variety of textures can be found, but not all of them are suitable for this task. The textures should be seamless, devoid of shadows, and free from any objects. Textures of floors, such as carpets, tiles, wood, concrete, and more, were utilized. Two approaches were employed to acquire the data for this thesis.

- Web Data Collection

  The data for this project was obtained from various online sources. Numerous free texture providers, such as textures.com, texturehaven.com, and others, were utilized for data acquisition. Due to the limitation of downloading one texture at a time from most websites, a series of scripts were developed to compile a list of suitable textures and automate the downloading process. These scripts were created using UiPath and Python.

- Video Game Textures

  The second approach involved using textures from video games. The advantage of this approach is that these textures are already seamless and often of high quality and quantity. However, a drawback is that these textures can be very repetitive. To obtain these textures, downward-facing recordings of the game were made, and the textures were extracted from the video. The major challenge with this approach is the need to disable shadows and all UI elements (HUD elements) in the game, which is not always possible.

1.3.2 Data Cleaning

To ensure that the data is consistent and free from elements that could corrupt the model, various cleaning steps were applied. For example, all images containing 3D objects were removed, especially those gathered from video games. During the recording of the floor, unwanted debris or pieces of wood were often present, and all extracted frames were manually checked.
In the case of web-gathered textures, there were different folder structures, and it was necessary to standardize them across all data folders. Additionally, some of them had associated files that were irrelevant to this use case and needed to be discarded.

All the images were in high-definition (HD) quality, with a height of approximately 1024 pixels.

Dataset	Size	Number of Images
FreePBR	452.0 MB	263
Polyhaven	298.0 MB	439
Poliigon	70.4 MB	49
Minecraft-Textures	636.0 MB	493
CsGoFloor-Textures	18.3 GB	44540
Combined	20.2 GB	45784

Table 1: Datasets collected for this thesis

1.3.3 Patterns in the data

To examine whether the dataset encompasses a broad spectrum of colors, multiple plots are created. These plots illustrate the color distribution within the datasets, providing insights into the diversity of colors present. Prior to plotting, a comprehensive pixel count across all images is conducted. For instance, if an image features 10 pixels of the color $(255, 0, 0)$, this count is added to a dictionary. Should the subsequent image in the dataset contain 5 pixels of the same color, these are also incorporated into the dictionary, cumulating a total of 15 for that specific color. This process is repeated for each color encountered, aggregating the counts to yield the overall color frequency within the dataset.

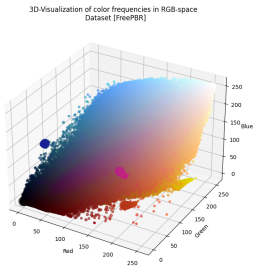
```
color_counts = {}
for i, (data, _) in enumerate(dataset):
    # data is a tensor of shape (3, height, width)
    pixel_rgb_array = (data.view(3, -1).t() * 255).to(torch.int32)

    for pixel_color in map(tuple, pixel_rgb_array):
        if color in color_counts:
            color_counts[pixel_color] += 1
        else:
            color_counts[pixel_color] = 1
```

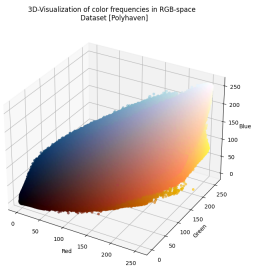
After analyzing the dataset through this method, visual representations of the color distributions were produced using Python and Matplotlib. These plots provide a three-dimensional view of the RGB color space, where the X, Y, and Z axes correspond to the Red, Green, and Blue color values, respectively, each ranging from 0 to 255.

$$\text{size} = \log(\text{count of color}) \times 20$$

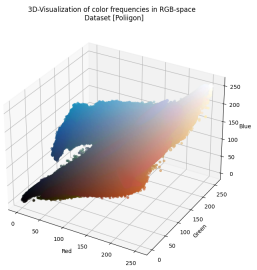
The size of each plotted point is calculated based on the logarithm of the color count, scaled by a factor of 20.



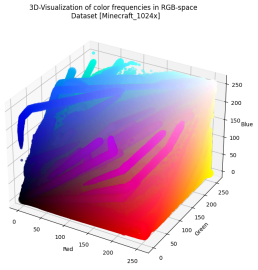
(a) FreePBR



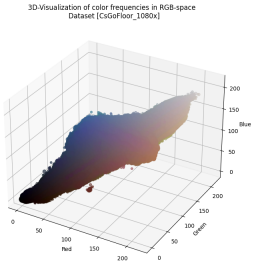
(b) Polyhaven



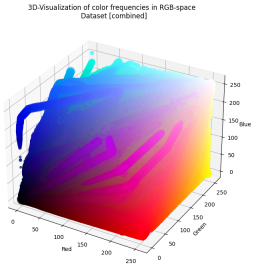
(c) Poliigon



(d) Minecraft-Textures



(e) CsGoFloor-Textures



(f) Combined

In the figure above, the color distributions of the individual datasets are shown. The first five subfigures represent the color distributions of the individual datasets, while the last subfigure (1f) shows the combined color distribution of all datasets. The color distributions of the individual datasets are quite similar, except for the Minecraft-Textures dataset, which is way more colorful than the others. The combined figure is a combination of all the individual datasets, and it is evident that the color distribution is quite diverse. This is a positive sign, as it indicates that the dataset is not focused on only a specific color spectrum.

1.3.4 Data Synchronization

In the thesis, a manual data synchronization routine is established to maintain data consistency between the supercomputer located in Berlin and the local workstation.

1.4 Training process

(gpu cluster göthingen, my GPU, ...)

1.5 Models

(LLMs, basic idea, roll model, spiral model)

1.5.1 Column Image Transformer

In the context of this thesis, a model termed the Column Image Transformer (CIT) has been conceptualized and developed. This model embodies an adaptation of the conventional transformer architecture. Distinctively, the CIT model diverges from traditional image processing techniques by segmenting the image into vertical slices or columns of pixels. This segmentation allows for a method where each column is processed on its own, following the "B" batch dimension in the model's structure.
The model assigns position embeddings to every pixel in a column, indicated by the "H" dimension. This approach enables the model to predict the properties of subsequent pixels within the same column, enhancing the model's ability to reconstruct image content and have a basic understanding of the context. Each pixel includes three values corresponding to the Red, Green, and Blue (RGB) color channels. These channels are processed across the model's layers as the "C" dimension.

1.5.2 Spiral Image Transformer

The second approach is represented by the Spiral Image Transformer (SIT). Unlike its predecessor, the Column Image Transformer (CIT), the SIT model employs a contextually spiral pattern. This architecture enables the generation of images starting from a central point and expanding outward (see ...). In the SIT model, the batch dimensions correspond to distinct images, whereas the H dimension represents the spiral context. Similar to the CIT model, the C dimension denotes the color channels.
One of the pivotal enhancements of the SIT model is its ability to analyze adjacent pixels on the horizontal axis, in contrast to the CIT model's limitation to columnar pixel analysis. This feature is particularly beneficial for interpreting textures with intricate patterns, such as diagonal ones, thereby offering an advantage over the Column Image Transformer. However, it is important to note that the SIT model operates within a constrained area of the image due to its 2D context. This limitation necessitates the use of only a portion of the image area, specifically a sector determined by the square root of the total area available to the Column Image Transformer (CIT), with an equal context length.

2 Experiment

2.1 Transformer behind the models

(Illustration, Tensor board)

2.1.1 evaluate the results

2.2 Roll model

(explanation, generating new content, , ,)

2.2.1 Classification or Regression

2.2.2

2.2.3 Discriminator

2.3 Spiral model

(explanation, Data to Spiral form, positional embedding,)

2.4 Problems

(layer norm(sigmoid vs clamp), color shift to gray (illustrations of average color), Text tokens vs imgs tokens)

3 Conclusion

3.1 Roll model

(strength, weaknesses)

3.2 Spiral model

(strength, weaknesses)

3.3 LLM Scaling Laws

3.4 Stable diffusion/ GANs with convolutional neural network

3.5 Further research

A Appendix

A.1 Unterabschnitt von Appendix

ABCDE

ABC

ABC

B Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht.
Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

(Ort, Datum, Unterschrift)

References

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., ... Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models.