IBM Innovation Center for Education

PHEMESOFT Software That Matters
TRUSTED DELIVERY PARTNER for IBM ICE

University of Petroleum and Energy Studies

Final Project Report

on

GENERATING REALISTIC TEXTURES FOR 3-D MODELS USING DEEP LEARNING

Team members:

Kushagra Singh - R214220646
Kushagra Singh - R2142201819
Kushagra Varshney - R2142201752
Kshitij Thakur - R214220632
Kshitij Nailwal – R2142201453

Guided by: Sumit Shukla
Industry Mentor: Sumit Shukla

Table of Contents

Executive Summary –

3D models are becoming increasingly popular in a variety of applications, such as gaming, animation, and virtual reality. However, one of the challenges of using 3D models is creating realistic textures.
Traditional methods for creating textures can be time-consuming and labor-intensive. They often require artists to manually create or paint the textures, which can be a tedious and repetitive process.
Deep learning offers a promising new approach to generating realistic textures for 3D models. Deep learning is a type of machine learning that uses artificial neural networks to learn from data. In the context of texture generation, deep learning can be used to learn the statistical distribution of real-world textures. This information can then be used to generate new textures that are similar to real-world textures.
There are a number of different deep learning methods that can be used to generate realistic textures for 3D models.

Objective:

Provide a definition of realistic textures and discuss their importance in 3D modeling. Develop a deep-learning model to develop textures for 3d models.

Approach:

Deep generative modeling has undergone a lot of recent research. Applications includinginpainting, conditional generation, and high-resolution picture synthesis have all made use ofGANs and VAEs. However, most works concentrate on the unrestricted or restricted productionof 2D images.The production of 3D geometry from latent codes or image collections has seen some effort in asimilar vein. However, there are not many works that use 3D data to direct image generation.Additionally, there are methods for creating innovative views that have mostly been studied inthe context of 2D conditional generative models. Such a representation is intrinsicallyconstrained because the 3D geometry of the scene or object for which a novel view needs to besynthesized is not explicitly known.

Model Training:

1. Collect a diverse and high-quality dataset of 2D texture images that represent the types of textures you want to generate in 3D. Ensure the dataset covers a range of materials (e.g., wood, stone, fabric) and variations (colors, patterns, details).
2. Apply data augmentation techniques (rotation, cropping, color adjustments) to increase the diversity of your dataset.
3. Choose an appropriate deep learning architecture for texture generation. Common choices include Convolutional Neural Networks (CNNs), Generative Adversarial Networks (GANs), or Variational Autoencoders (VAEs).

4. Create a mapping between 2D texture images and 3D model UV coordinates. This mapping will be crucial for applying the generated textures to the 3D models correctly.
5. Generator Loss: Typically includes content loss (e.g., mean squared error) and adversarial loss (e.g., binary cross-entropy) to ensure both realism and style in the generated textures.
6. Discriminator Loss: Encourages the discriminator to distinguish between real and generated textures.
7. Train your deep learning model using the training dataset and defined loss functions.
8. Monitor training progress using loss metrics and visual inspection of generated textures.
9. Fine-tune the generated textures to match specific lighting and environmental conditions within your 3D scene.
10. Apply post-processing techniques to generated textures if needed. This may include adjusting colors, enhancing details, or ensuring tileability.

Challenges:

- Obtaining a diverse and high-quality dataset of 2D texture images can be challenging. High-resolution, clean, and well-labeled textures are essential for training.
- Training deep learning models for 3D texture generation is computationally demanding and may require access to powerful GPUs and substantial training time.
- Deep learning models can easily overfit to the training data, resulting in textures that lack generalization to new scenarios or lighting conditions. Careful regularization and validation are necessary to mitigate this issue.
- Designing effective loss functions that balance realism, style, and texture diversity can be challenging. Finding the right trade-offs between these aspects is not always straightforward.
- Ensuring that generated textures are tileable (i.e., can be repeated seamlessly) is essential for many applications. Achieving perfect tileability can be challenging and often requires post-processing.
- Training data may have biases in terms of material types, lighting conditions, or object geometries. These biases can lead to unrealistic textures when applied to different scenarios.
- Integrating the texture generation model into a 3D rendering pipeline or application can be complex, especially when dealing with various file formats and software libraries.

1. BACKGROUND

1.1 Aim

The intended audience for this project is but not limited to Individuals interested in 3D models or for recreational purposes might want to learn about generative adversarial network (GAN). This audience would be interested in experimenting with and implementing these algorithms. Students studying 3D modeling, artificial intelligence, or related fields would benefit from learning about GAN. Educators teaching courses or conducting workshops on these topics would also be part of this audience.

1.2 Technologies

The list of technologies used to generate realistic images are:

1. Tensorboard
2. Collab

1.3 Hardware Architecture

- High-Performance GPU
- Sufficient RAM
- CPU: Multi-core processor

1.4 Software Architecture

- Deep Learning Framework:TensorFlow or PyTorch

2. SYSTEM

2.1 Requirements

2.1.1 System Requirements

- The algorithm should be compatible with a range of GPUs, CPUs, and memory configurations commonly used for Deep Learning tasks.
- Windows Operating System.
- High-Performance GPU with T4 accelerator.
- RAM 8Gb and above.
- CPU: Multi-core processor.

2.1.2 User Requirements

- Input image should be 640x640 pixels.
- Image should be related to buildings only.
- The images should be of good resolution.

2.1.3 Environmental Requirements

- Python version 3.0 and above.
- TensorFlow version 2.0 and above.
- Keras version 2.0 and above.
- NumPy version 1.2 and above.
- Image Processing Libraries: OpenCV Framework version 4.2 and above.

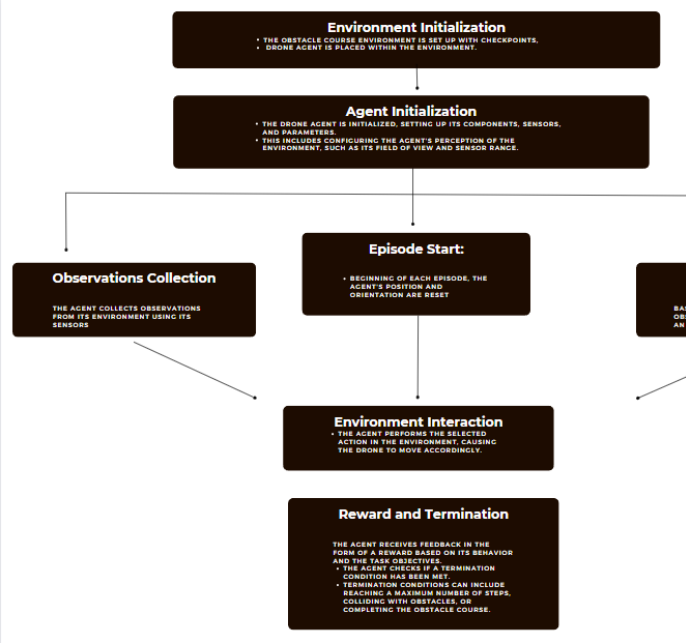
2.2 Desing and Architecture

A multi-step procedure is required to create realistic textures for 3D models using GAN in order to develop high-quality and varied textures. First, to ensure consistency, a varied array of real-world textures
is gathered and preprocessed.. The models for the generator and discriminator are created, with the generator responsible for synthesizing realistic textures and the discriminator for determining the

difference between actual and fake textures. Using cutting-edge methods like mini-batch discrimination or spectral normalization, the GAN learns how to enhance texture production and stabilize. Tuning hyperparameters is essential for maximizing GAN performance. To evaluate texture quality, human judgements and objective metrics are also used. Once taught, the generator produces a variety of texturesthat can be used with 3D models. Iterative refinement and user feedback can improve the GAN. Finally, theGAN is deployed for texture generation in applications like games, virtual reality, and architecturalvisualizations. Computationally intensive, GANs require substantial resources, but their iterative natureallows for continuous improvement and better results.

It initiateswith a simple data loader method where we load the dataset which contains various images of textures ofreal life things. We train our model using this dataset with various parameters such as batch size, learningrate, no. of epochs, optimizers and lastly the loss. Loss plays a vital role in our model. We use the crossentropy loss function for the discriminator. The generator and discriminator consists of various neuralnetworks with different layers such as Convolutional Layer, ReLU etc. The loss that is calculated by thediscriminator us propagated back to reduce error. This process continues until we get optimum results andreduced loss.

```
participant Agent : AircraftAgent
participant Area : AircraftArea
participant Unity ML-Agents : ML-Agents Framework

Agent->Unity ML-Agents: Initialize ML-Agents environment
Unity ML-Agents->Area: Initialize Area
Unity ML-Agents->Agent: Initialize Agent

Agent->Agent: Initialize episode
Agent->Area: ResetAgentPosition(agent, randomize)

loop Until training completion or termination conditions
    Agent->Agent: CollectObservations(sensor)
    Agent->Agent: Request actions from RL algorithm
    Agent->Agent: OnActionReceived(vectorAction)
    Agent->Area: Update agent's position and orientation

    alt Training Mode
        Agent->Area: Calculate rewards based on agent's behavior
        Agent->Unity ML-Agents: AddReward(reward)
        Agent->Unity ML-Agents: EndEpisode() if termination conditions met
    end

    Agent->Agent: Check for checkpoint reached
    Agent->Agent: Check for timeout condition

    Agent->Agent: Increment step count
    Agent->Agent: Check if max step count reached
end

Unity ML-Agents->Unity ML-Agents: Update model parameters

Agent->Unity ML-Agents: Finalize training process
Unity ML-Agents->Agent: Disable learning
Unity ML-Agents->Area: Reset environment

Agent->Unity ML-Agents: Shutdown ML-Agents environment
```

1. 

.

2.4 Testing

2.4.1 Test Plan Objectives

The objectives of the test plan are to:

- Verify that the model can successfully generate textures from random noise or latent vectors..
- Ensure the generated textures align with the intended style or material type.
- Provide confidence that the system is ready for deployment.

2.4.2 Data Selection

Gather diverse and representative real-world texture images. Include various materials, patterns, and colors..

2.4.3 Security

Deep learning models, including those for texture generation, are susceptible to adversarial attacks. Evaluate the robustness of your model against attacks that attempt to manipulate or deceive it.

2.4.4 Test Strategy

Verify that your training data is clean, consistent, and wellpre-processed.Check for missing data, duplicate textures, or incorrectly labeled samples.Measure the time it takes for the model to generate textures and ensure it meets real-time requirements if applicable.Monitor latency when generating textures in batch mode.

2.4.5 System Test

System testing for a deep learning model used in 3D texture generation involves assessing the entire system, including the model, its deployment environment, and its interaction with other components.

2.4.6 Performance Test

Push the system to its limits to identify potential points of failure or weaknesses. This could include extreme inputs, excessive concurrent requests, or resource exhaustion scenarios.

2.4.7 Security Test

Conduct a security assessment to identify vulnerabilities and ensure that the system is resilient to attacks. This includes checking for data leaks, unauthorized access, and code vulnerabilities.

2.4.8 System Integration Testing

Verify the integration of the deep learning model into the broader system. Test how it interacts with other components, such as 3D rendering engines or user interfaces.

The following are some of the key objectives of SIT:

- To ensure that the individual components of the system can communicate with each other effectively.
- To identify and fix any integration issues that may exist between the different components of the system.
- To verify that the system meets its requirements in terms of functionality, performance, and reliability.

2.4.9 Acceptance Testing

Develop test cases based on the acceptance criteria. These test cases should be designed to evaluate the model's functionality and performance as it relates to the users' requirements..Execute functional test cases to verify that the model meets the specified functional requirements. This may include testing the model's ability to generate textures of a specific style or material.

2.4.10 Conclusion

In conclusion, the development and deployment of a deep learning model for 3D texture generation is a complex and multifaceted endeavor. It involves a series of critical steps, from data collection and model training to testing, deployment, and ongoing maintenance.

2.5 Graphical User Interface (GUI) Layout

Designing the graphical user interface (GUI) for a project involving a deep learning model for 3D texture generation is crucial for user interaction and control.Display real-time previews of the generated textures as the model processes the inputs.Implement zoom and pan controls for users to inspect textures closely. If you have multiple models or variations, offer a dropdown or selection menu for users to choose which model to use. Implement logging features to capture logs and diagnostic information for troubleshooting and debugging purposes. Include a debug mode for advanced users or developers. Include an option to upload custom models if applicable.

2.6.2 Static Code Analysis

Static code analysis is an essential practice in software development, including projects involving deep learning models for 3D texture generation. It helps identify code issues, potential vulnerabilities, and maintainability concerns before the code is executed.

The static code analysis tool will be configured to look for potential defects in the following areas:

- Memory management
- Control flow
- Data flow
- Threading
- Security

The results of the static code analysis will be used to fix any defects that are found.

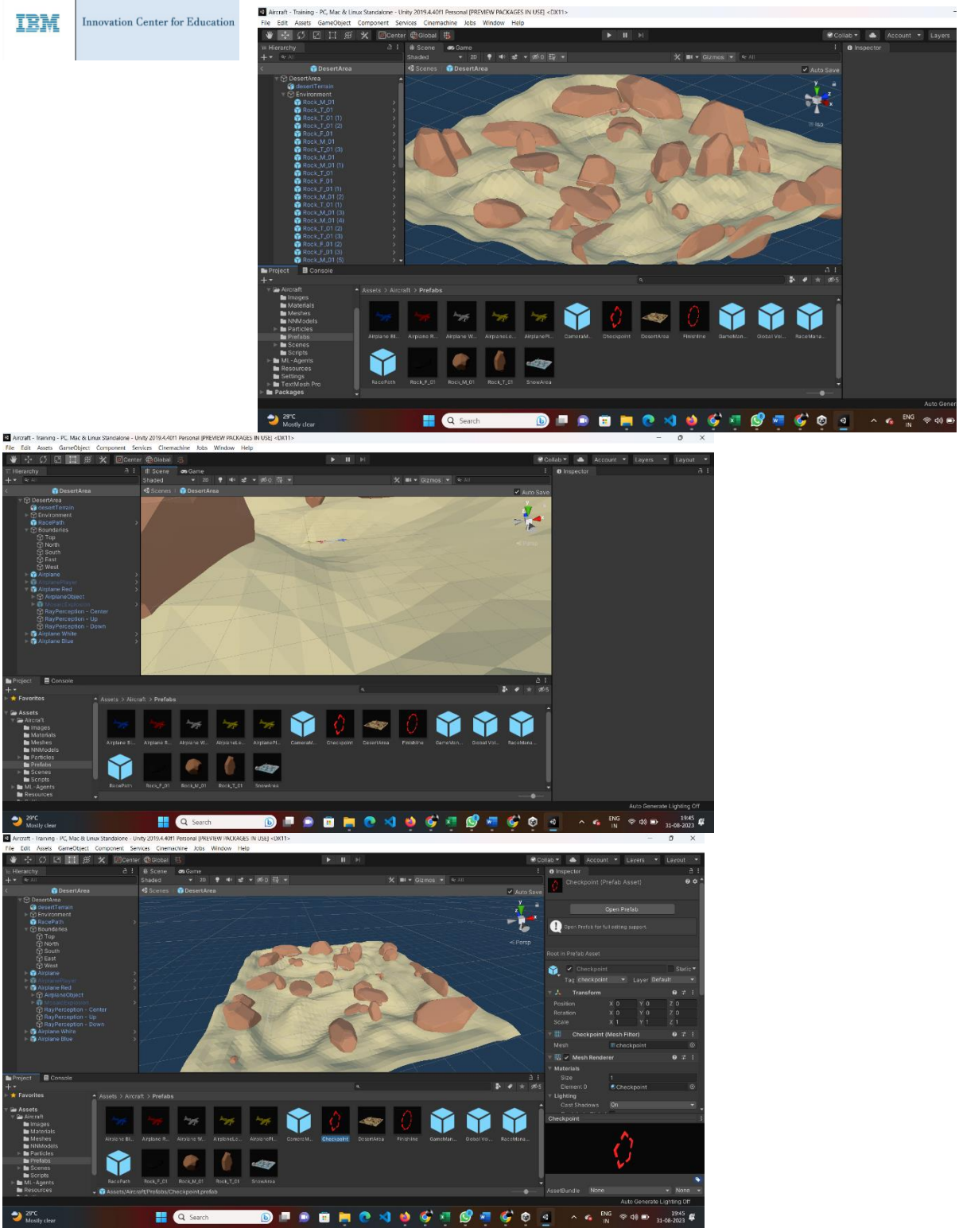
2.6.3 Test of Main Function

Use static analysis tools to identify code segments that could lead to performance bottlenecks. This is particularly important if your project has real-time requirements.Unit testing will be used to test the main function in isolation. System testing will be used to test the main function in the context of the entire system. User acceptance testing will be used to test the main function with users.

The results of the testing will be used to ensure that the main function meets the system's requirements.

3. **Snapshots of the Project**

The following are snapshots of the drone obstacle avoidance system:



4. Conclusions

The recent advances in deep learning for texture generation have shown that this technology has the potential to revolutionize the 3D modeling industry. Deep learning can be used to generate realistic textures for a wide variety of objects, including animals, plants, and inanimate objects. This can be used to create more realistic and immersive 3D experiences in gaming, animation, and virtual reality.

However, there are still some challenges that need to be addressed before deep learning can be widely used for texture generation. One challenge is that deep learning models can be computationally expensive to train. Another challenge is that deep learning models can be biased, which can lead to the generation of unrealistic textures.

5. References

[1] M.Cimpoi, S. Maji, and A. Vedaldi. Deep convolutional filter banks for texture recognitionand segmentation. arXiv:1411.6836 [cs], November 2014. arXiv: 1411.6836.

[2] Y. Zhang, W. Chen, H. Ling, J. Gao, Y. Zhang, A. Torralba, and S. Fidler. Image gansmeet differentiable rendering for inverse graphics and interpretable 3d neural rendering.International Conference on Learning Representations, 2020.

[3] Leon Gatys, Alexander Ecker, and Matthias Bethge. Texture synthesis using convolutionalneural networks. In Advances in Neural Information Processing Systems 28, 2015.

[4] Lucas Theis and Matthias Bethge. Generative image modeling using spatial LSTMs. InAdvances in Neural Information Processing Systems 28, 2015

6 Appendix

The appendix contains additional information about the drone obstacle avoidance system, such as the system's architecture and the code used to implement the system.