INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, NAGPUR

(An Institution of National Importance by Act of Parliament)

Computer Science and Engineering Department

Topics in AI

2024-2025

Report on

Ray tracing denoising using Blender & Deep Learning

Submitted by-

Name	**Enrolment Number**
Ved Vartak	BT21CSE177
Abhishek Kumar	BT21CSE188
Shreyash Suradkar	BT21CSE105
Nitin Kumar	BT21CSE112

Submitted to-

Dr Nishat Ansari

Assistant Professor

ABSTRACT:

This project addresses the rise in parallel computational capabilities, ray-tracing is becoming more prevalent in computer graphics, enabling highly realistic lighting and shadows. For real-time rendering, reducing samples per pixel and utilizing deep neural networks for denoising allows us to achieve visually accurate results while maintaining performance, making ray-tracing more accessible in gaming and interactive applications.

INTRODUCTION:

This project explores the implementation of an image denoising model using deep learning techniques. Image denoising is a crucial preprocessing step in computer vision applications, aiming to remove noise from images while preserving key features. By utilizing tools such as TensorFlow and NumPy, this project trains a neural network to restore clarity in noisy images, enhancing their quality for subsequent use in real-world applications.

OBJECTIVE:

The objective of this project is to develop a U-Net-based model and dataset creation using blender tool . The model will be trained to identify and remove various types of noise while retaining essential image features. The project aims to achieve high accuracy and generalization, contributing to advancements in noise reduction for tasks such as image restoration, enhancement, and preprocessing for machine learning pipelines. It also aims of creation of specific scenario of the world view using blender tool and by placing different camera angle towards various object and create a database.

Caustics are a well-known source of noise, causing Fireflies. They happen because the renderer has difficulty finding specular highlights viewed through a soft glossy or diffuse reflection. There is a No Caustics option to disable glossy behind a diffuse reflection entirely. Many renderers will typically disable caustics by default.

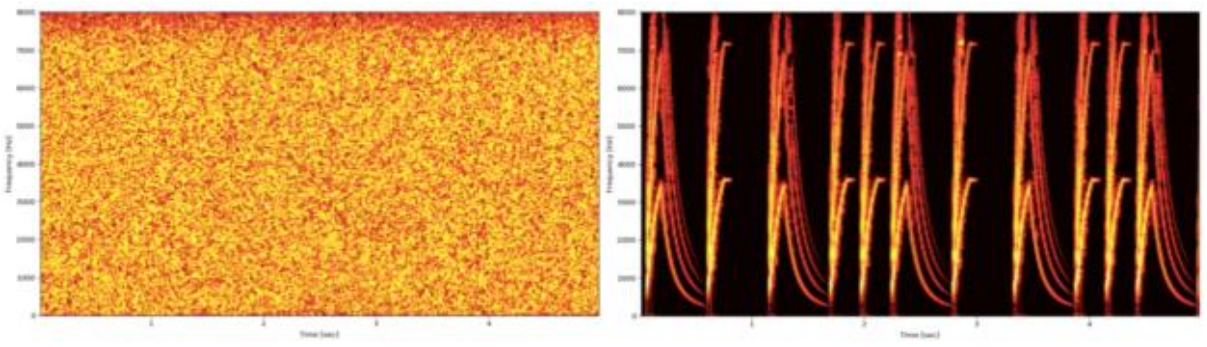
APPLICATIONS:

- Medical Imaging: Deep learning-based noise reduction enhances the quality of MRI, CT, and X-ray scans by removing artifacts and noise, leading to clearer images for accurate diagnosis.
- Photography and Videography: In low-light or high-ISO conditions, deep learning models can reduce noise in photos and videos, improving image quality without losing detail.
- Surveillance Systems: Noise reduction in security camera footage improves clarity, especially in low-light environments, making it easier to identify objects or individuals.
- Astronomy: Deep learning is used to clean noisy astronomical images captured by telescopes, helping scientists detect faint objects and details in space

SECTION 2: LITERATURE REVIEW

1. Real-Time Noise Suppression Using Deep Learning

- **Overview**: NVIDIA's Real-Time Noise Suppression (RTNS) leverages AI to enhance voice quality by reducing background noise in real-time, targeting applications like video conferencing and live streaming.
- **Model**: It uses a deep learning-based framework built on the U-Net architecture, optimized for NVIDIA GPUs to deliver low-latency performance.
- **Performance**: The model achieves superior noise suppression while maintaining



natural voice quality, outperforming traditional DSP-based techniques.
- **Results**: Demonstrated significant improvements in audio clarity across diverse noise scenarios, making it highly effective for real-world applications.
- **Conclusion**: NVIDIA's RTNS offers a robust, scalable solution for real-time communication, setting a new standard for AI-driven audio enhancement.
-

2. Intel Open Image Denoise:

- **Overview:** Open Image Denoise is an Intel-developed library for high-performance, AI-powered denoising of rendered images, targeting ray tracing applications.
- **Model:** It utilizes deep learning models optimized for Intel architectures, particularly leveraging the capabilities of CPUs for seamless integration.
- **Performance:** The library delivers high-quality denoising, preserving fine details, textures, and sharpness in rendered images across a variety of scenes.
- **Results:** Benchmarks show excellent performance and quality, making it a preferred choice for film production, architectural visualization, and gaming.
- **Conclusion:** Open Image Denoise is a powerful, open-source tool that enhances rendered image quality, supporting professional and creative workflows.

-

3. GenAI in Blender: Transforming Image Production :

- **Overview:** The paper discusses enhancing Blender's functionality using Generative AI technology to empower artists.
- **Model Name:** Utilizes Generative Adversarial Networks (GANs) like StyleGANs and Stable Diffusion.
- **Methodology:** Developed a user-friendly add-on with a prompt-based architecture for seamless integration of AI-generated content.
- **Result:** High-quality, realistic images generated with vivid details and consistency using AI models.
- **Conclusion:** The integration of Generative AI in Blender revolutionizes artistic expression, enabling artists to effortlessly incorporate AI-generated content into their workflows

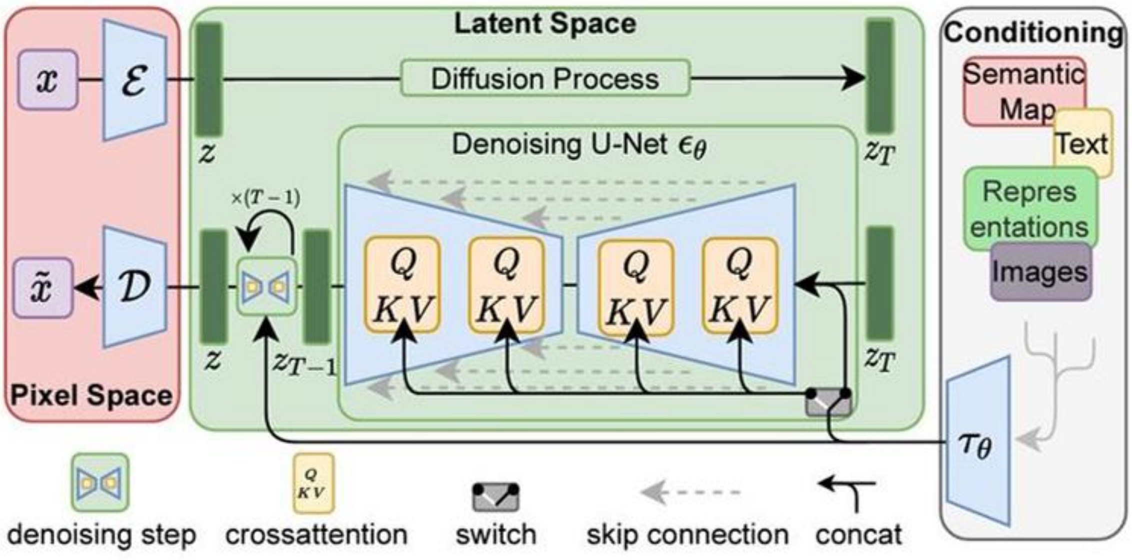
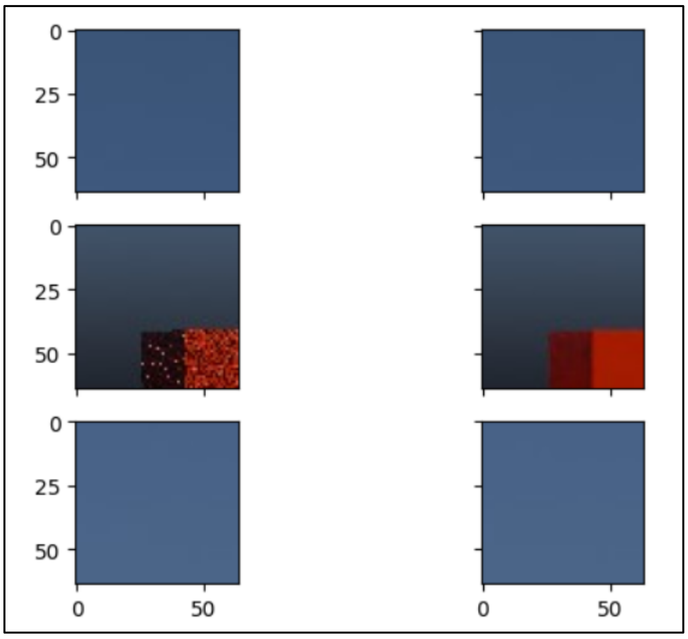


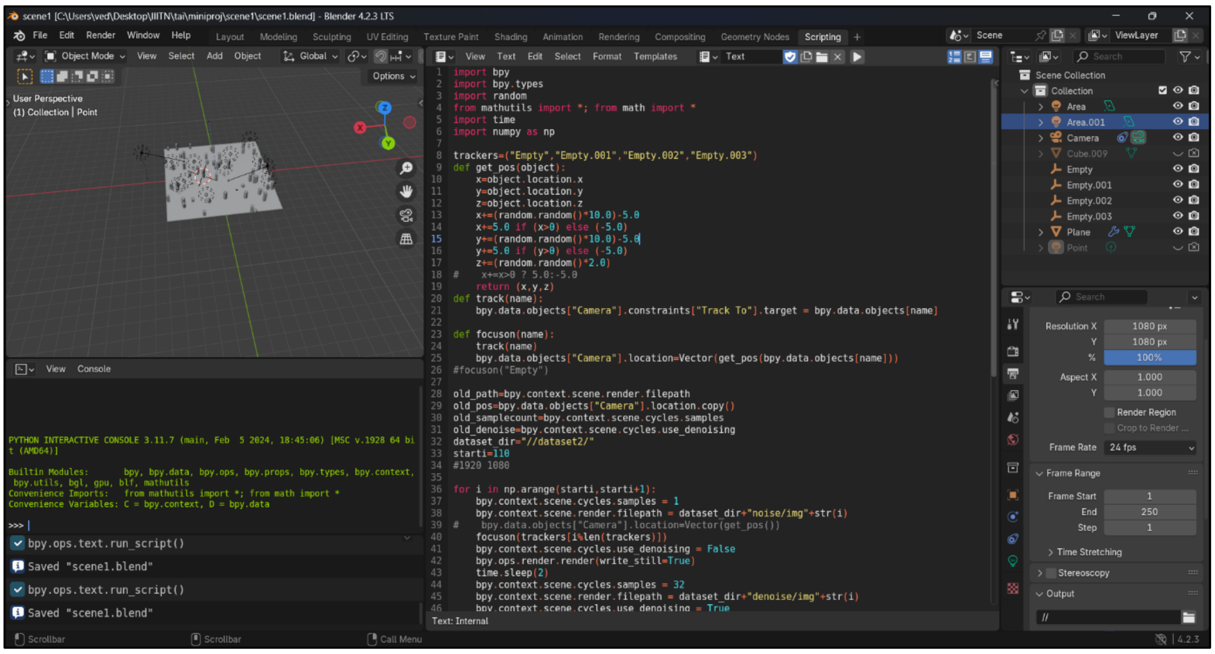
Figure 2: Latent Diffusion Model (Source: Reference [3])

SECTION 3: METHODOLOGY:

1) Data Creation:

- To create dataset for our denoiser we used Blender to render 3D Models at different sampling rate (Ex: For noisy Data Sample 1spp and for Denoised Dataset 1024spp)
- Blender is a free and open-source 3D creation suite that supports modelling, sculpting, animation, rendering, video editing, and VFX.
- We created scene with a plane and placed cubes on it at random position and random orientation with random material using Geometry Shaders.
- To generate Renders we used Blenders scripting API to place camera at random position for every image sample.



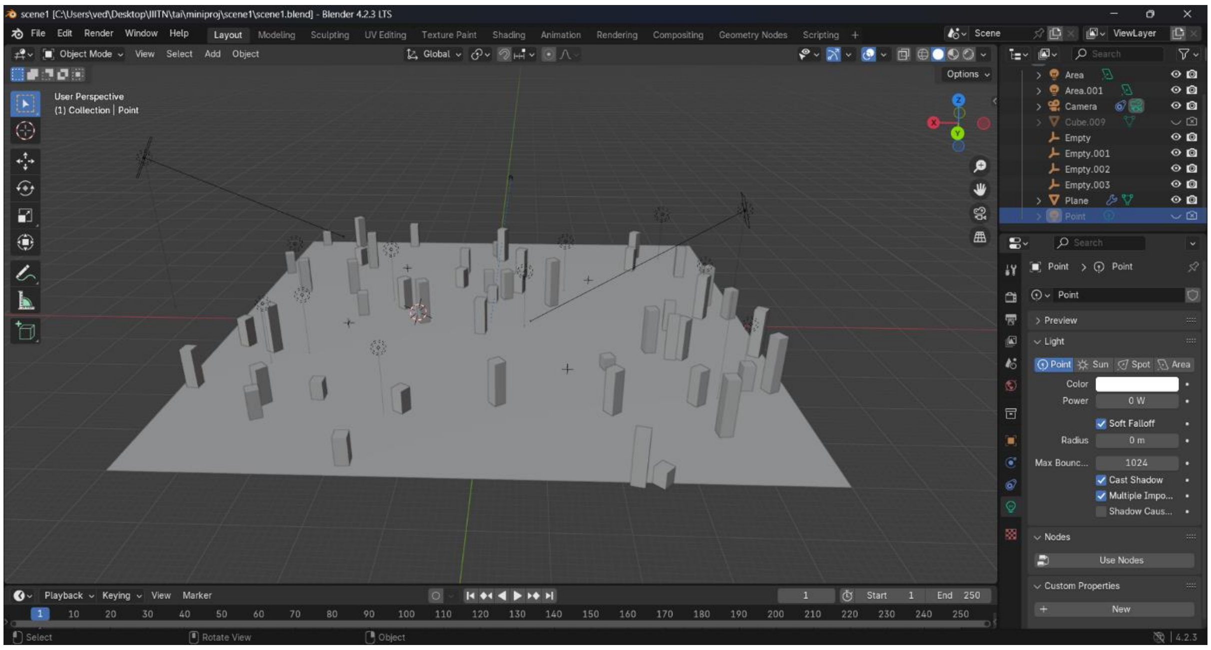


scene1 [C:\Users\ved\Desktop\IIITN\tai\miniproj\scene1\scene1.blend] - Blender 4.2.3 LTS
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes Scripting
Object Mode View Select Add Object Global
User Perspective
(1) Collection | Point
View Text Edit Select Format Templates Text
Scene ViewLayer
Scene Collection
Collection
Area
Area.001
Camera
Cube.009
Empty
Empty.001
Empty.002
Empty.003
Plane
Point
View Console
Text: Internal
Scrollbar Scrollbar Call Menu

```python
import bpy
import bpy.types
import random
from mathutils import *; from math import *
import time
import numpy as np

trackers=("Empty","Empty.001","Empty.002","Empty.003")
def get_pos(object):
    x=object.location.x
    y=object.location.y
    z=object.location.z
    x+=(random.random()*10.0)-5.0
    x+=5.0 if (x>0) else (-5.0)
    y+=(random.random()*10.0)-5.0
    y+=5.0 if (y>0) else (-5.0)
    z+=(random.random()*2.0)
#    x+=x>0 ? 5.0:-5.0
    return (x,y,z)
def track(name):
    bpy.data.objects["Camera"].constraints["Track To"].target = bpy.data.objects[name]

def focuson(name):
    track(name)
    bpy.data.objects["Camera"].location=Vector(get_pos(bpy.data.objects[name]))
#focuson("Empty")

old_path=bpy.context.scene.render.filepath
old_pos=bpy.data.objects["Camera"].location.copy()
old_samplecount=bpy.context.scene.cycles.samples
old_denoise=bpy.context.scene.cycles.use_denoising
dataset_dir="//dataset2/"
starti=110
#1920 1080

for i in np.arange(starti,starti+1):
    bpy.context.scene.cycles.samples = 1
    bpy.context.scene.render.filepath = dataset_dir+"noise/img"+str(i)
#    bpy.data.objects["Camera"].location=Vector(get_pos())
    focuson(trackers[i%len(trackers)])
    bpy.context.scene.cycles.use_denoising = False
    bpy.ops.render.render(write_still=True)
    time.sleep(2)
    bpy.context.scene.cycles.samples = 32
    bpy.context.scene.render.filepath = dataset_dir+"denoise/img"+str(i)
    bpy.context.scene.cycles.use_denoising = True
```

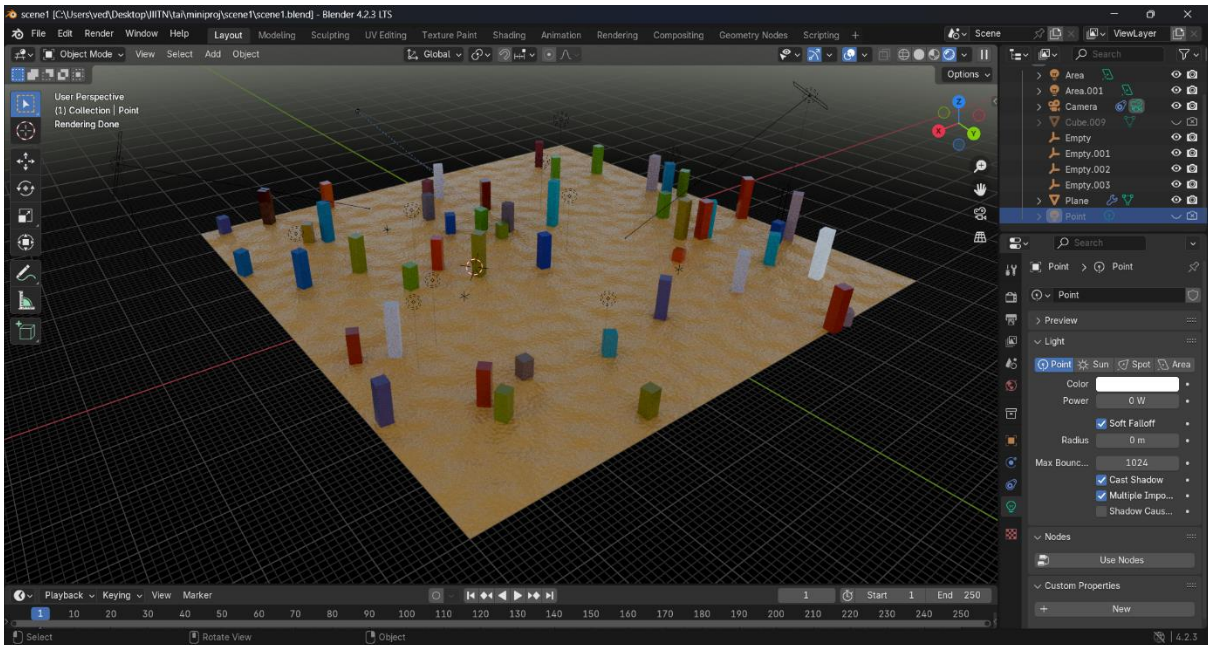
```
PYTHON INTERACTIVE CONSOLE 3.11.7 (main, Feb  5 2024, 18:45:06) [MSC v.1928 64 bi
t (AMD64)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context,
 bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>> 
bpy.ops.text.run_script()
Saved "scene1.blend"
bpy.ops.text.run_script()
Saved "scene1.blend"
```

```
Resolution X   1080 px
Y              1080 px
%              100%
Aspect X       1.000
Y              1.000
Render Region
Crop to Render ...
Frame Rate     24 fps
Frame Range
Frame Start    1
End            250
Step           1
Time Stretching
Stereoscopy
Output
//
```


scene1 [C:\Users\ved\Desktop\IIITN\tai\miniproj\scene1\scene1.blend] - Blender 4.2.3 LTS

User Perspective
(1) Collection | Point

Light: Point
Color
Power 0 W
Soft Falloff ✓
Radius 0 m
Max Bounc... 1024
Cast Shadow ✓
Multiple Impo... ✓
Shadow Caus...


scene1 [C:\Users\ved\Desktop\IIITN\tai\miniproj\scene1\scene1.blend] - Blender 4.2.3 LTS

User Perspective
(1) Collection | Point
Rendering Done

Light: Point
Color
Power 0 W
Soft Falloff ✓
Radius 0 m
Max Bounc... 1024
Cast Shadow ✓
Multiple Impo... ✓
Shadow Caus...

2) Data Collection & Preprocessing:

Data preprocessing involves cleaning and transforming raw data into a format that can be easily understood and used by machine learning algorithms.

File Details

- **File**: dataset2.zip
- Contains:
    - A directory of noisy images (noise) for input data.
    - A directory of clean, denoised images (denoise) for ground truth labels.

Image Preprocessing

- **Image Shape**: All images are resized or cropped to 64x64x3 (height, width, and colour channels) to ensure consistency in input dimensions for the model.
- **Scaling**:
  - Pixel values are normalized to the range [0, 1] for numerical stability during training.
  - The helper function PIL_to_data() converts images into this normalized format.

Data Loader Classes

1) DataLoader:

- Handles the loading of noisy and denoised image pairs from their respective directories.
- **Cropping**:
  - A fixed cropping region ensures consistent extraction of image patches.
- **Usage**:
  - The load() function takes two parameters: offset (starting index) and n (number of images to load).
  - Outputs:
    x: Array of noisy images (input data).
    y: Array of corresponding denoised images (ground truth).

2) RanCropDataLoader (Random Crop Data Loader):

- Inherits from DataLoader and extends its functionality to generate random image patches.
- **Random Patch Extraction**:
  - For each image, patches_per_img random patches are extracted.
  - Random cropping coordinates ensure diversity in training data.
- Outputs are normalized noisy and denoised patches as x and y.

3) Model Architecture :

3.1 Regression Models

The **Modified UNet** model is a convolutional neural network (CNN) designed with an **encoder-decoder structure**, making it ideal for image-to-image translation tasks such as segmentation and denoising. In this case, UNet is utilized to reconstruct clean, denoised images from noisy inputs.

Below is a comprehensive explanation of the UNet model, training methodology, and its application to image denoising.

1. Architecture

1. **Encoder (Downsampling Path)**:
    - The encoder progressively reduces the spatial dimensions of the input image while extracting hierarchical features.
    - Composed of multiple blocks (encoder1 to encoder4), each block consists of:
        - **Two 3×3 Convolutional Layers**: These layers apply kernels to the input, capturing spatial features.
        - **Batch Normalization**: Normalizes activations, reducing internal covariate shift and improving training stability.
        - **ReLU Activation**: Introduces non-linearity to enable the model to learn complex patterns.
    - Each block is followed by a **Max Pooling** layer, which halves the spatial dimensions (downsampling).
2. **Bottleneck**:
    - The bottleneck acts as a bridge between the encoder and decoder.
    - It captures high-level features with a higher number of filters (features × 16) while maintaining the smallest spatial dimensions.
3. **Decoder (Upsampling Path)**:
    - The decoder reconstructs the spatial resolution of the input while utilizing features extracted by the encoder.
    - Composed of:
        - **Transposed Convolutional Layers (Upconvs)**: Double the spatial dimensions, performing learnable upsampling.
        - **Concatenation with Encoder Features (Skip Connections)**: Merges encoder features with decoder activations to preserve spatial details and improve reconstruction.
        - **Two Convolutional Layers per Block**: Similar to the encoder, these refine the upsampled features.
4. **Output Layer**:
    - A Conv2D layer with a kernel size of 1 and a **sigmoid activation function** outputs the final reconstructed image with pixel values in the range [0, 1].

4) Model Training:

Advanced Denoising Strategies

The provided code demonstrates three different strategies for denoising images using a trained model: **Direct Denoising**, **Grid-Based Denoising**, and **Sliding Window Denoising**. These approaches handle the limitations imposed by model input sizes and the need to process images of varying dimensions.

1. Base Denoising Class

The Denoiser class serves as the base class for all denoising strategies:

- **Attributes**:
    - model: The trained UNet model used for denoising.
    - patch_shape: The shape of patches accepted by the model, derived from the model's image_shape attribute.
- **Method**:
    - denoise(image): The placeholder function for image denoising. In this base class, it simply returns the input image.

2. Grid-Based Denoising

The GridDenoiser class processes the input image in a grid-like manner:

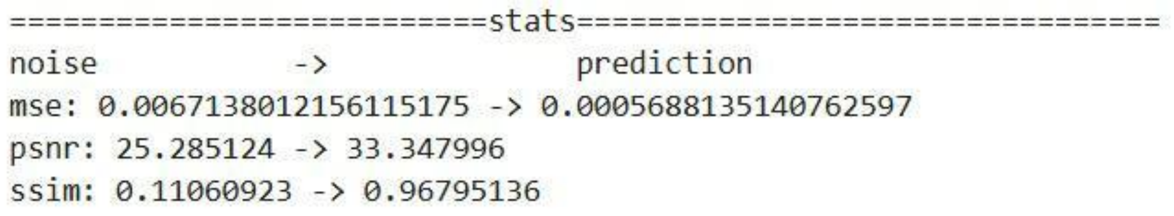
- **Steps**:
    1. **Padding**: Pads the image dimensions so they are divisible by the patch size (patch_shape). This ensures the entire image can be divided into non-overlapping patches.
    2. **Patch Extraction**: Divides the padded image into grid-aligned patches of size patch_shape.
    3. **Batch Prediction**: Feeds all patches into the model at once and collects predictions.
    4. **Patch Reconstruction**: Reassembles the denoised patches back into the original image dimensions, cropping the padded areas.
- **Advantages**:
    1. Efficient for fixed-size patches and images with regular grid-aligned dimensions.
- **Limitations**:
    1. Doesn't handle overlapping patches, which may lead to boundary artifacts between patches.

3. Sliding Window Denoising

The SlidingWindowDenoiser class implements a sliding window approach to process the image:

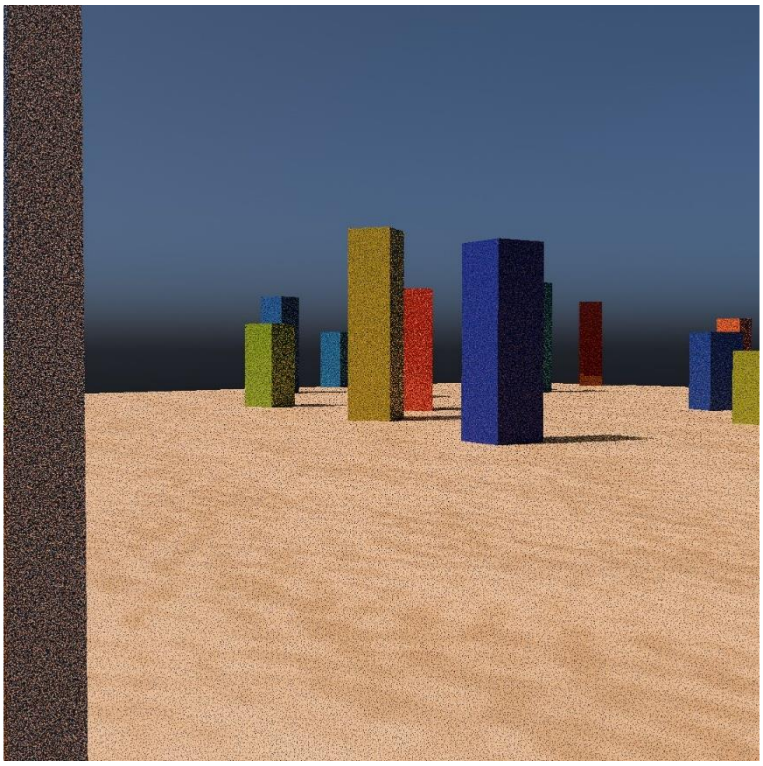
- **Steps**:
    1. **Padding**: Pads the image to ensure sliding windows can cover the entire image.
    2. **Sliding Window Extraction**: Iteratively extracts overlapping patches using a configurable stride (slide).
    3. **Batch Prediction**: Processes patches in batches to leverage GPU parallelism.
    4. **Patch Aggregation**: Combines overlapping patches by averaging their contributions.
    5. **Cropping**: Crops out the padding to restore the original image dimensions.
- **Advantages**:
    1. Handles overlapping patches, which reduces boundary artifacts and improves denoising quality.
    2. Configurable stride allows tuning the balance between computational cost and reconstruction quality.
- **Limitations**:
    1. Slightly slower than grid-based denoising due to overlap processing and averaging.

SECTION 4: Results & Conclusion

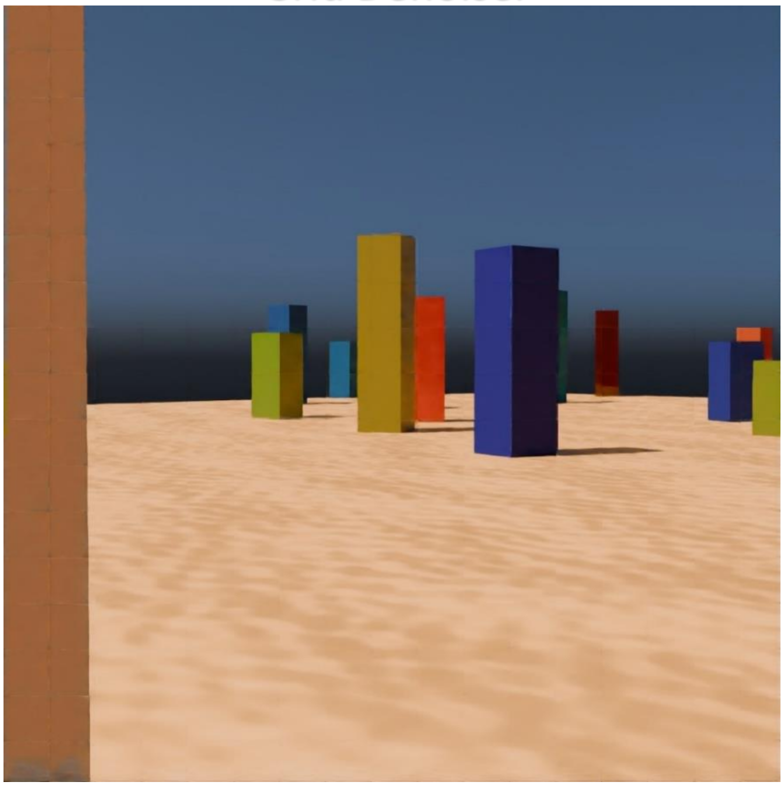


```
=============================stats===================================
noise           ->               prediction
mse: 0.0067138012156115175 -> 0.0005688135140762597
psnr: 25.285124 -> 33.347996
ssim: 0.11060923 -> 0.96795136
```

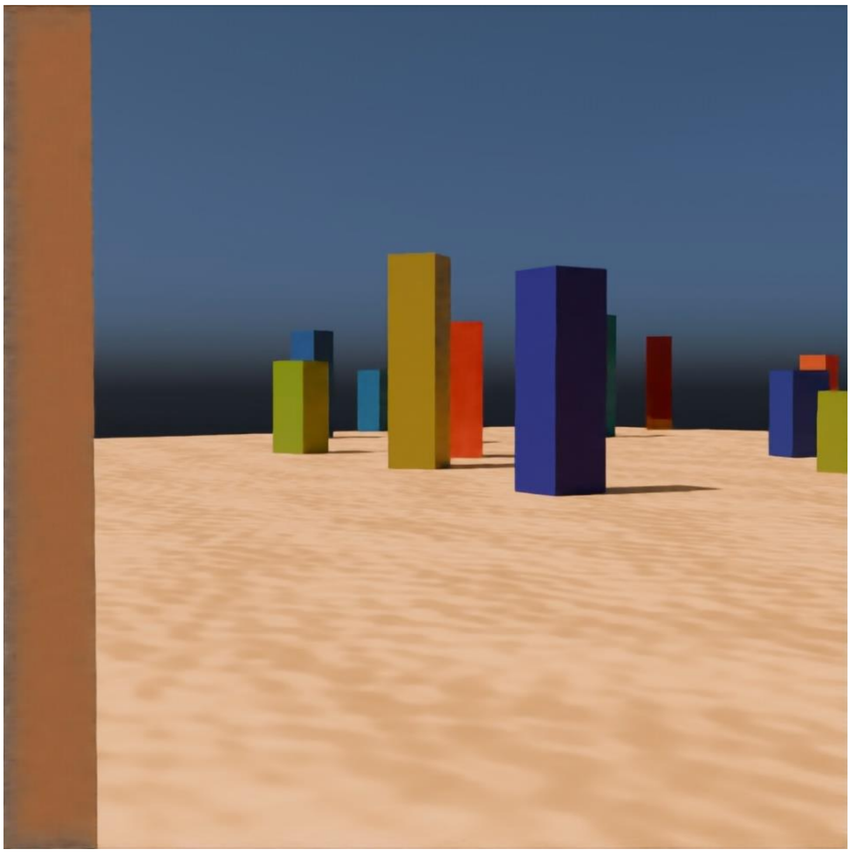
Noise



Grid Denoiser



Sliding Window



Sliding Window x2