Multi-Resolution Neural Materials (NeuMIP): Methodology and Rendering

Yuxuan Zhu

Advisor: Professor Shuang Zhao

Department of Computer Science

July 2023

1. Introduction

Multi-resolution neural material (NeuMIP) is a neural method for rendering various material appearances at different scales proposed by Alexander Kuznetsov et al. from the University of California, San Diego. It applies the pyramids of neural textures for mipmapping, combined with a fully connected network as a material-specific multi-layer perceptron (MLP) decoder. Notably, the introduction of neural offsets enables intricate parallax effects without the need for tessellations.

Conventionally, material models have relied on complex mathematical equations to mimic light absorption, scattering, and reflection on different surfaces. Consider the Bidirectional Reflectance Distribution Function (BRDF) model, which utilizes the reflectance equation to predict the path of the light direction. Oren-Nayar model, for instance, which accounts for the roughness of surfaces and provides a more accurate representation of diffuse reflection, can render material with a shadowing effect. However, these traditional models face challenges in reproducing the intricacies of more complex materials and parallax effects, leading to long time processing and storage constraints.

In contrast to the limitations of the traditional material reflectance models, NeuMIP offers several advantages, including better efficiency, adaptability, and realism. NeuMIP combines the neural networks with structured pyramidal organization to optimize computational efficiency, resulting in faster rendering times and reduced storage requirements. Neural networks in NeuMIP can learn and adapt from extensive datasets, allowing the model to generalize and precisely replicate a wide array of material appearances across levels of detail. Moreover, by introducing neural offsets, NeuMIP allows the generation of intricate parallax effects, which are often a challenge for conventional models. This leads to materials that display depth and visual complexity, enhancing overall realism.

In this work, we provide a simplified explanation of the algorithm of Multi-resolution bidirectional texture function (MBTF) and neural MBTF with Neural Offset. Lastly, my duplicated experimental results are presented, demonstrating the effectiveness of the NeuMIP method in generating material appearances.

2. Background

This section introduces the basic concept of Bidirectional texture function (BTF) and multi-resolution BTF as the fundamental background of the NeuMIP method. The algorithm, application, and limitation of each function are discussed.

2.1 Bidirectional Texture Function

Bidirectional texture function (BTF) describes the appearance of texture as a function of illumination and viewing direction. At a coarse scale, where local surface variations are subpixel and local intensity is uniform, the Bidirectional Reflectance Distribution Function (BRDF) characterizes the material's appearance, while at a fine scale, where surface variations result in local intensity variations, BTF is utilized (Dana 1999). The BTF, similar to the BRDF, is a 6D function that takes into account 2D location coordinates, incoming direction, and outgoing direction to determine the reflectance value. Despite its effectiveness, one drawback of the BTF is its large memory requirement, which is solved by many new methods today.

2.2 Multi-resolution BTF

Kuznetsov's group has proposed a revision by introducing an extra input, the radius of the footprint kernel σ. The function M is defined by:

$$M(\mathbf{u}, \sigma, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) = \int_{\mathbb{R}^2} G(\mathbf{u}, \sigma; \mathbf{x}) B(\mathbf{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})\, d\mathbf{x},$$

Where G is a normalized 2D Gaussian function (which describes a bell-shaped curve centered around the means) and B is the traditional BTF we discussed above. In this approach, the MBTF value is obtained by calculating the weighted average of the radiance in the outcoming direction from a light source with unit irradiance. Note this definition aligns with the definition of BRDF which represents the outgoing radiance per unit of incoming irradiance. While the introduction of weighting improves the performance of the MBTF compared to the traditional BTF, MBTF is still not perfect. According to Kuznetsov's team, the 7D function MBTF can be computationally prohibitive to store directly. Additionally, Kuznetsov's team has also identified redundancies within the MBTF function, particularly in certain materials.

3. NeuMIP Method

In this section, we introduce the NeuMIP method, commencing with the foundational baseline neural MBTF featuring a neural texture pyramid and MLP decoder. Subsequently, we delve into the neural MBTF with neural offsets (NeuMIP) and elaborate its pipeline. Lastly, we encompass a detailed breakdown of the algorithm's three steps: the neural offset module, the neural texture pyramid, and the MLP decoder.

3.1 Neural MBTF Baseline

To revise MBTF, Kuznetsov's team proposed a baseline neural MBTF that can already work for many complex materials. The baseline contains a neural texture pyramid P and a material decoder network F. They will be explained in detail in the next section. The neural MBTF is expressed by:

$$M(\mathbf{u}, \sigma, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) = F(P(\mathbf{u}, \sigma), \boldsymbol{\omega_i}, \boldsymbol{\omega_o}),$$

As the neural texture pyramid is added, neural textures could express spatial information, encoding complex microgeometry effects. Then, a lightweight MLP efficiently decodes the texture and enables

faster evaluation, reducing the redundancy problem in MBTF. However, in some cases, such as highly non-flat material with a significant parallax effect, the neural MBTF baseline will be difficult to handle.

3.2 Neural MBTF with Neural Offset and Pipeline

The full neural MBTF representation introduces a novel method, the neural offset module, to compute the new lookup location. The full neural MBTF is described as

$$M(\mathbf{u}, \sigma, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) = F(P(\mathbf{u} + O(\mathbf{u}, \boldsymbol{\omega_o}), \sigma), \boldsymbol{\omega_i}, \boldsymbol{\omega_o}).$$

In this context, the symbols P and F still represent a neural feature lookup from a neural pyramid P and an MLP decoder network F, respectively. O is the neural offset module function.

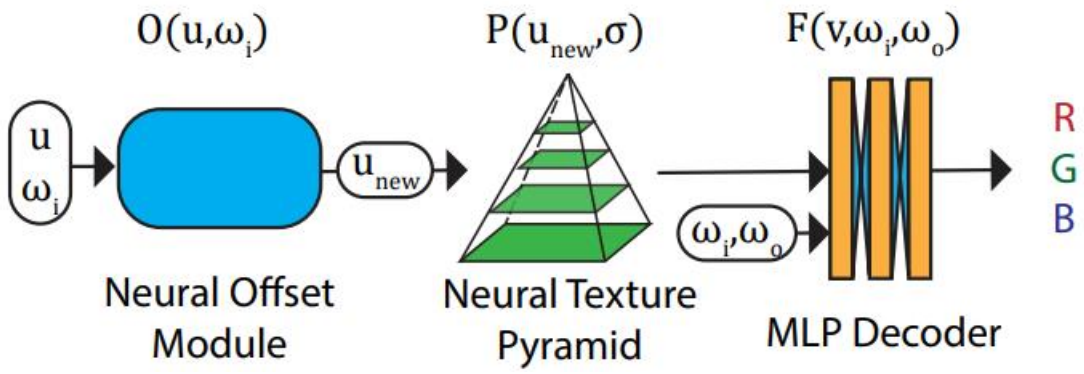

Fig.1. NeuMIP method's pipeline (Kuznetsov 2021)

Figure 1 shows the pipeline of the neural MBTF with neural offset. Given the 2d location coordinates u and light incoming direction ωi, the neural offset module predicts the new location in coarse geometry. The new location is passed into the neural texture pyramid, generating a 7-channel feature vector using trilinear interpolation. Lastly, the feature vector, light incoming and outcoming direction are fed into the MLP decoder to predict the RGB reflectance value as the final color of the image.

3.3 Neural Offset Module

Kaneko et al. in 2001 discussed shape representation with parallax mapping, analyzing the calculation of texture coordinate shift and depth approximation. This idea becomes one of the inspirations for the neural offset module.

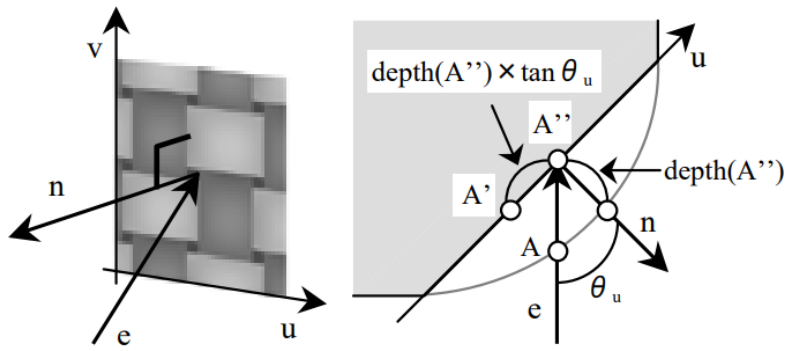

Fig.2. The geometry of texture coordinate shift (Kaneko 2001)

The left-hand side of Figure 2 shows the coordinate system of the curved surface defined by the u, v axes, and e and n are the observation direction and surface normal. In the right-hand side of Figure 2, given location A to find new location A'', we can express the translation of the shifted texture coordinates as u ’= u + tan(θu) × depth(u,v), where u and u’ are the source and resulting location coordinate of u axis. The coordinate of the v-axis can be obtained by the same formula. Unlike this classic method of parallax rendering, the neural offset module presents through unsupervised learning and does not require an input normal. Furthermore, the normals of the supporting materials are also not explicitly delineated.

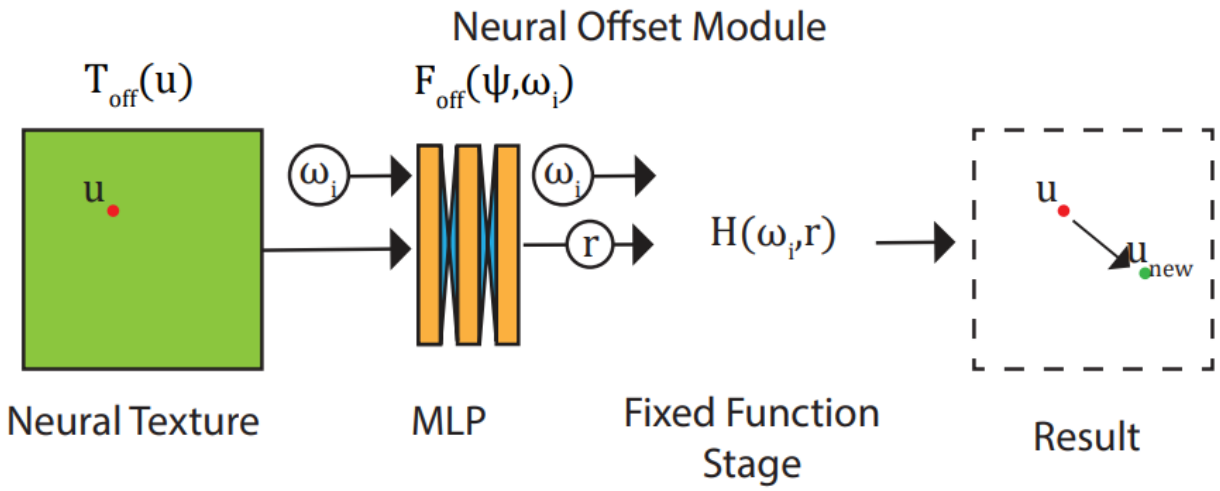

Fig.3. Neural offset module (Kuznetsov 2021)

The neural offset module is a network that predicts a coordinate lookup for the feature texture lookup, applied to improve the image quality in the complex parallax effect. As shown in Figure 3, the module consists of three components: a neural offset texture Toff, an MLP decoder Foff that regresses the ray depth r from Toff, and a fixed function H that outputs the result.

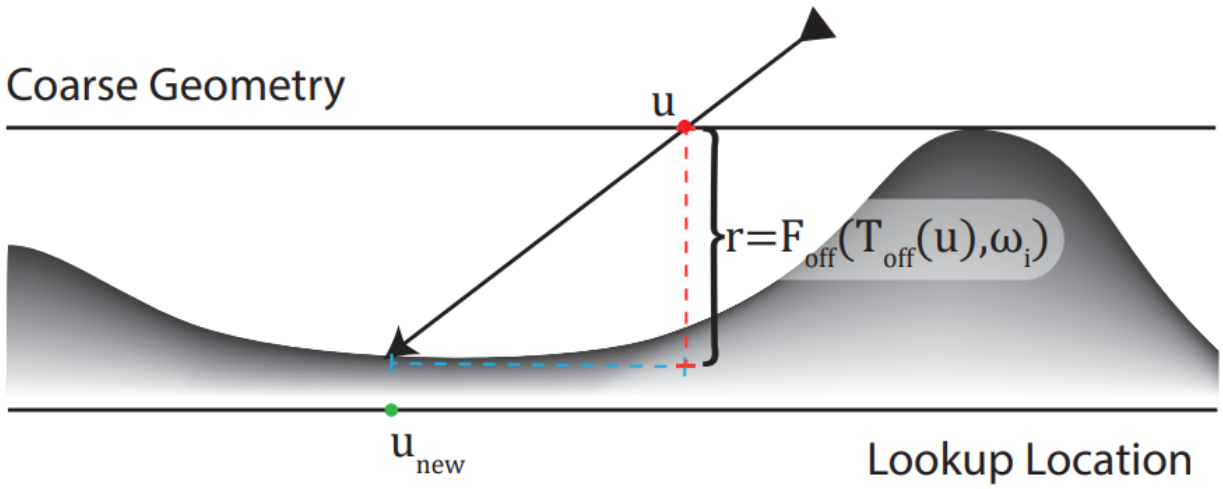

Fig.4. Fixed function stage (Kuznetsov 2021)

Kuznetsov et al. trained the network to regress a 1D scalar value ray depth and then turned it into a 2D offset given the view direction, easing the regression task. While the neural texture Toff is the texture information at location u, the MLP decoder is utilized to predict ray depth r in the texture lookup by a bilinear interpolation (see Figure 4). The ray depth is computed by:

$$r = F_{\text{off}}(\boldsymbol{\psi}, \boldsymbol{\omega_o}) = F_{\text{off}}(T_{\text{off}}(\mathbf{u}), \boldsymbol{\omega_o}),$$

Where $\boldsymbol{\psi}$ is the latent feature vector lookup in Toff on the initial location u. The MLP decoder takes the latent feature vector and the viewing direction as input, generating result r by a fully connected function that consists of 4 layers and outputs 25 channels in each layer (except the last one). The code of the ReLU activation function can be implemented as:

```
class FullyConnected1(torch.nn.Module):
    def __init__(self, num_in, num_out=3):
        super(FullyConnected1, self).__init__()
        self.num_in = num_in
        self.num_out = num_out

        self.func = torch.nn.Sequential(
            torch.nn.Conv2d(num_in, 25, 1),
            torch.nn.ReLU(),
            torch.nn.Conv2d(25, 25, 1),
            torch.nn.ReLU(),
            torch.nn.Conv2d(25, 25, 1),
            torch.nn.ReLU(),
            torch.nn.Conv2d(25, self.num_out, 1),
        )
```

As we find the ray depth, now the new location unew can be calculated by:

$$\mathbf{u}_{\text{new}} = \mathbf{u} + H(r, \boldsymbol{\omega_o}) = \mathbf{u} + \frac{r}{\max(\boldsymbol{\omega}_z, .6)}(\boldsymbol{\omega}_x, \boldsymbol{\omega}_y).$$

Note H is the fixed function and ωx, ωy, and ωz are the components of viewing direction ωo.

3.4 Neural Texture Pyramid

In 1983, an innovative approach referred to as the "Pyramidal Parametric" was proposed by Lance Williams. The feature of the pyramidal data structure is a succession of levels that vary the resolution at which the data is represented. The image plane is subdivided into quadrants recursively until the subsequent analysis of a given subsection indicates that the complexity of surface order has been sufficiently reduced to facilitate rendering. Mipmapping is a particular format for two-dimensional parametric functions, offering greater speed than texturing algorithms.

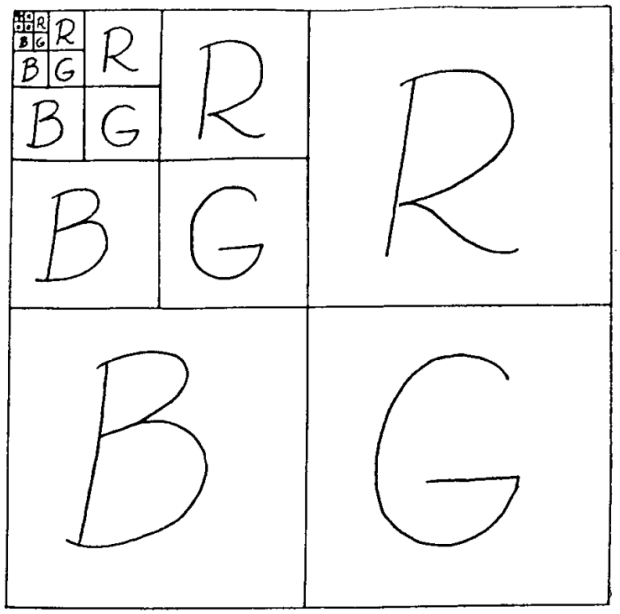

Fig.5. color mipmapping (Williams 1983)

Figure 5 illustrates the memory structure of a color mip map. The image is separated into red, green, and blue components (as R, G, and B in the figure). Successively filtered and down-sampled versions of each component are instanced above and to the left of the originals, resulting in a succession of increasingly smaller images. Each image in this series has half the linear dimension of its parent image and is located above and to the left of its predecessor. The corresponding points in the prefiltered maps can be accessed by executing a binary shift on an input U, V coordinate pair. Since the filtering and sampling procedures are performed at scales of powers of two, this enables the indexing of these maps through cost-effective binary scaling operations.

In contrast, the neural texture pyramid applies the neural texture, leveraging a textured pyramid P = {Ts} consisting of a set of neural texture Ts, which is a bilinearly-interpolated neural texture lookup from the level s. Each Ts has a size of 2 ^ s * 2 ^ s * c (number of channels), where s denotes a discrete level of details and lets s greater or equal to 0 and less or equal to some positive integer k. The pyramid structure can effectively encode complex material appearance at multiple scales. The mipmapping procedure is similar to standard color mipmapping but the pre-level neural textures are independent from one another. Each neural texture within the set is individually optimized. This independent optimization serves to guarantee that the MBTF is capable of effectively representing the appearance of the material across all levels. Utilizing trilinear interpolation, we procure a neural feature denoted as P (u, σ) at a specific location u and at a continuous level σ, where u is the spatial dimensions of the location

coordinate obtained by bilinear interpolation, followed by linear interpolation in the logarithmic space for the scale dimension. The neural feature v at location u calculated by the neural offset at level σ is computed by:

$$\boldsymbol{v} = P(\mathbf{u}, \sigma) = w_1 T_{\lfloor l \rfloor}(\mathbf{u}) + w_2 T_{\lceil l \rceil}(\mathbf{u})$$ where l = log2(σ), $w1$ = ($\lceil l \rceil - l$), $w2$ = ($l - \lfloor l \rfloor$).

3.5 MLP Decoder

A multilayer perceptron (MLP) is a class of feedforward artificial neural networks. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Each node (except input nodes) represents a neuron that employs a nonlinear activation function. The supervised learning method that MLP employs for its training process is known as backpropagation (Mohanty 2019). MLP generates output sequences from the learned representation of input (the output of the encoder). In the case of NeuMIP, the MLP decoder's functionality is similar to a BRDF function: find RGB values by input location, view, and light direction.

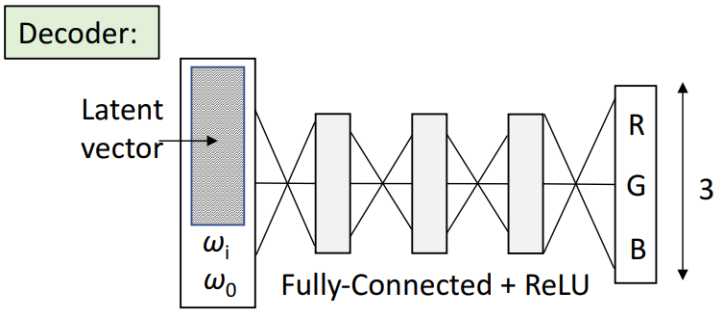

Fig.6. MLP decoder architecture (Rainer et al. 2020)

As shown in Figure 6, the MLP decoder concatenates the latent vector with the view and light direction and passes through four hidden fully connected layers with ReLU activations (same functions in the neural offset module), and finally outputs the RGB values. More specifically, the input of the network has c + 4 values, where c values come from the neural texture and 4 comes from the 2D view and light direction vectors. Each intermediate layer has 25 output channels, except the last one, which has 3 channels presented as the final RGB reflectance values of the material. To construct models that accurately capture materials with locally intricate microgeometry, the neural textures express the spatial information, encoding complex microgeometry effects. Therefore, a lightweight MLP is appropriate to decode the texture with fast evaluation.

4. Results

My rendering results are shown in this section. The image comparisons are provided, along with my rendering methods, analysis, and reflection of the result.

4.1 Flat Image Rendering

In my duplication of the material rendering, the neural network dataset is trained and the evaluate function is applied given certain input values of a light, view direction, and level σ to generate a flat 2D image in the form of an RGB NumPy array. Several parameters employed in image generation align with those utilized by the authors. The three types of material images are shown below.

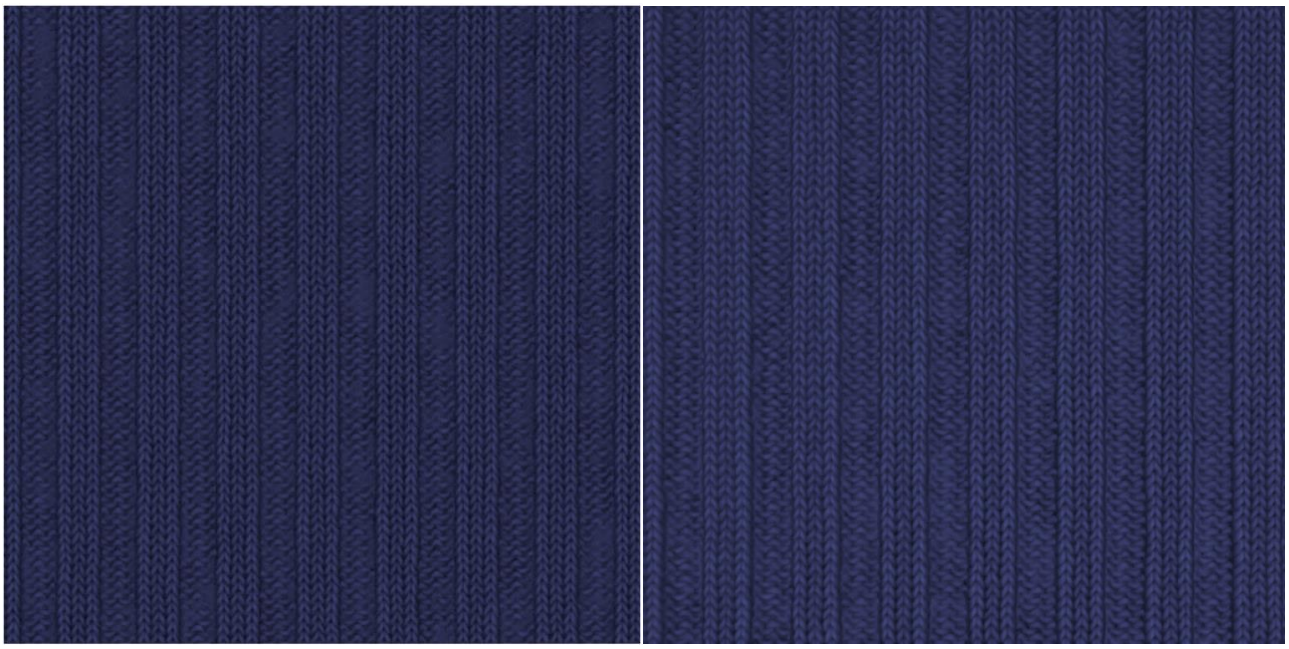

Fig.7. material wool (straight). Left: duplication. Right: Kuznetsov et al.

Fig.8. material wool2 (twisted).

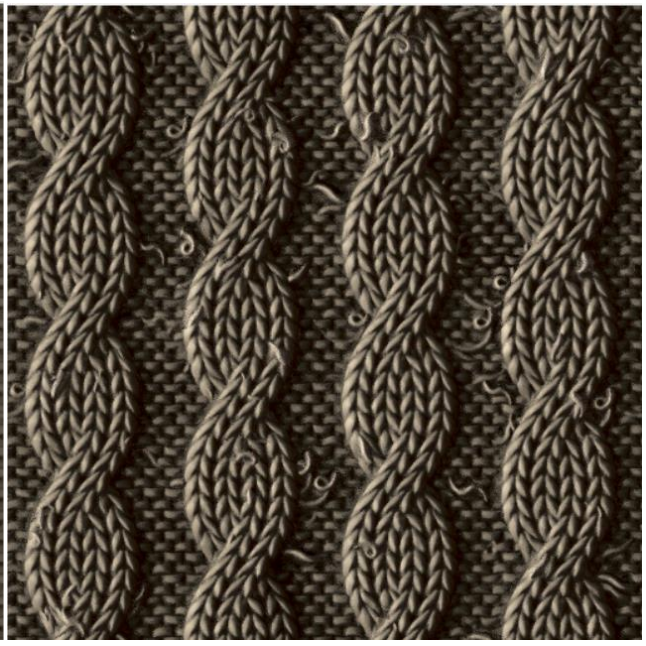

Fig.9. parallax effect at light position (-1, 0).

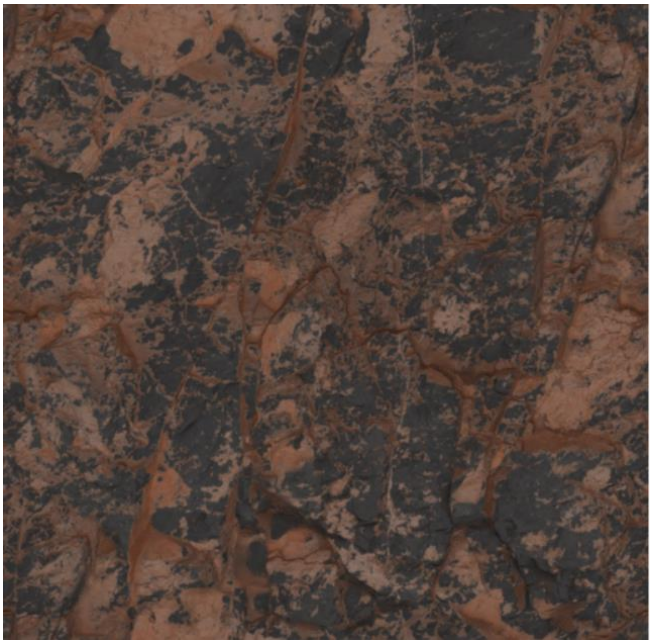

Fig.10. material cliff rock.

In Figure 7, the left-hand side presents my replications of NeuMIP utilizing the wool dataset, while the right-hand side depicts the corresponding images from the WebGL real-time demonstration conducted by Kuznetsov's team. Figures 8-10 produce other materials of wool2 and cliff datasets. For the images in Figures 7, 8, and 10, the light source and viewing locations are both fixed at coordinates (0, 0). In Figure 9, we altered the position of the light source to (-1, 0) to show the parallax effect. As the light source is positioned on the left-hand side (where x = -1), the shadow shifts to the right of the twisted texture. This observation corresponds with the comparison images as well as the real world.

4.2 Curve (sphere) Image Rendering

First, we created a sphere shape mesh grid using the linspace and meshgrid functions from the NumPy library. The coordinates of points on the sphere can be calculated by the trigonometric function sin and cos. Since the sphere is built centered at the origin, the normal vector is equal to the position vector of each point. As we discussed above, the NeuMIP function works for the local coordinate system, which means the normal coordinate is fixed at (0, 0, 1). Therefore, to apply the evaluate function, the location, light direction, and view direction coordinates are required to be transformed from the world coordinate system to the local coordinate system. In this case, we utilized a camera class with a project function and a to-local function to convert the points.

```
class Camera:
    def __init__(self, f, c, R, t):
        self.f = f #focal length float
        self.c = c #offset of principle point 2*1
        self.R = R #camera position 3*3
        self.t = t #camera translation 3*1

    def project(self, pts3):
        pts3_cam = self.R.T @ (pts3.reshape(-1, 1) - self.t.reshape(-1, 1))
        pts2_cam = self.f * pts3_cam[:2, :] / pts3_cam[2, :]
```

```
        pts2 = pts2_cam + self.c.reshape(-1, 1)
        return pts2
```

The class camera records the focal length(f), the principal point offset(c), rotation, and translation vector(r & t) and applies the parameters to the 3D point. By transforming, rotating, and scaling to return the 2D point under a certain camera, the project function will be called for each location coordinate on the sphere to return the 2D local coordinate.

```
def to_local(wi, wo, n):
    x = np.array([n[1], -n[0], 0]) #x axis
    x = x / np.linalg.norm(x) if np.linalg.norm(x) != 0 else x # normalize
    y = np.cross(n, x) #y axis
    transform = np.array([x, y, n])
    local_wi = np.dot(wi, transform)
    local_wo = np.dot(wo, transform)
    return local_wi[:2], local_wo[:2] #return 2d
```

The to_local function converts the light direction (wi) and view direction (wo) coordinates from a 3D to a 2D local coordinate system. We can consider the normal as the z-axis unit vector. Then, the x and y axis unit vectors can be found by the properties of the dot and cross product. Lastly, the local coordinates of light and view direction are obtained by calculating a dot product to the transform vector.

After processing in the evaluate function, the resulting image is shown as:

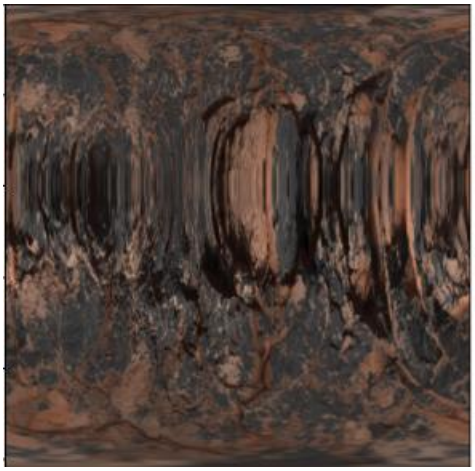

Fig.11. intermediate image from the evaluate function (at the light source (0, 0, -3)).

The last step is mipmapping the image to the 3D sphere. This work is done by using the OpenGL in PyGame library, generating a sphere and mipmap on it. Figure 12 illustrates the final results with three light sources at (-2, 0, -3), (0, 0, -3), and (2, 0, -3). In the case of the middle image, the light source is directed toward the sphere's center. The normal at the sphere's center is parallel to the direction of light, creating an absence of shadow at the center but an evident shadow around the edges. On the other hand, the image with the light source at the coordinate (2, 0, -3), for example, presents that the light source direction is not orthogonal to the surface normal vector at the sphere's center. As a result, we observe a parallax effect even at the sphere's center. The three images with different light source directions have different shadow areas accordingly.

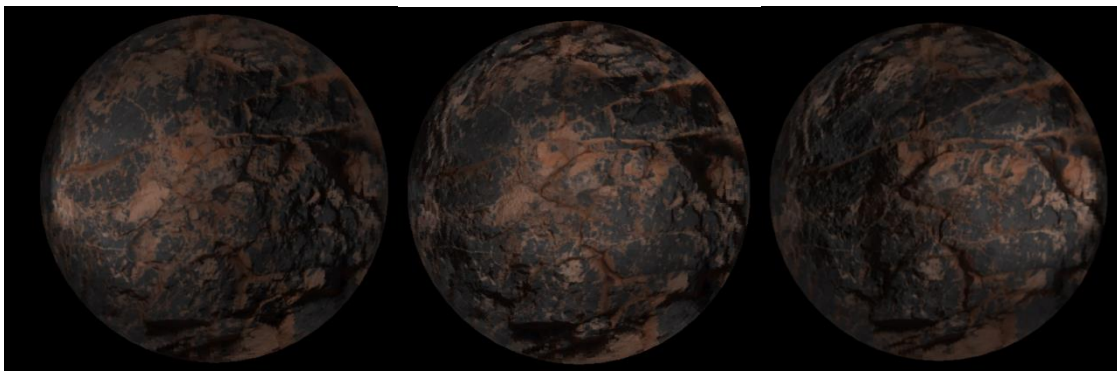

Fig.12. Final result of the sphere rendering. Material: cliff. Left: light source at (-2, 0, -3). Middle: light source at (0, 0, -3). Right: light source at (2, 0, -3).

Fig.13. Kuznetsov et al. Left: light source at (-2, 0, -3). Middle: light source at (0, 0, -3). Right: light source at (2, 0, -3).

Figure 13 shows the comparison to the NeuMIP authors' work. Admittedly, there exists a certain degree of visual discrepancy between the results from my work and their work. The reasons come from using different parameters, such as the setting of diffuse and ambient light, light intensity, number of points on the sphere, and size of the texture image. It is conceivable that the different rendering techniques employed might also account for these differences. Since the source code of the WebGL demo is not available to the public, it is difficult to achieve perfect visual consistency between the sets of images. Despite the differences, my work is valid in the aspects of cliff texture rendering and shadow positioning.

5. Conclusion

NeuMIP is a fast-computing material rendering method invented by Kuznetsov's team. The neural architecture learns the 7D MBTF function. In the neural MBTF pipeline, we apply the neural offset module to find the new location under the surface, the neural texture pyramid to generate a 7D channel feature vector, and finally, the MLP decoder to obtain RGB reflectance values. In my work, the techniques are duplicated, and flat and curved images are rendered.

However, the NeuMIP technique has some limitations including the implementation of simple importance sampling and the difficulty of handling some very specular materials. In Kuznetsov's team's

latest work, instead of training on synthetic mesostructures applied to infinite planes, they trained the neural representation on a dataset to learn the correct behavior across surfaces with varying curvatures. They also built upon the NeuMIP architecture by considering curvature as part of the material query and transparency as the query output. In this case, the new method can generate better-quality curved images.

6. Acknowledgments

Special thanks to Professor Zhao for his continued support and guidance in my understanding of the paper, code implementation, and computer graphics learning. Thanks to Kuznetsov et al. for their pioneering work, which served as my primary learning reference for my HP project.

References

Dana, K., Ginneken, B., Nayar, S. K., & Koenderink, J. (1999). *Reflectance and texture of real-world surfaces*. ACM Transactions on Graphics (TOG), 18(1), 1-34.

Kaneko, T., Takahei, T., Inami, M., Kawakami, N., Yanagida, Y., Maeda, T., & Tachi, S. (2001). *Detailed shape representation with parallax mapping*. ICAT 2001.

Kuznetsov, A., Mullia, K., Xu, Z., Hašan, M., & Ramamoorthi, R. (2021). *NeuMIP: Multi-resolution neural materials*. Transactions on Graphics (Proceedings of SIGGRAPH), 40(4), Article 175.

Kuznetsov, A., Wang, X., Mullia, K., Luan, F., Xu, Z., Hašan, M., & Ramamoorthi, R. (2022). *Rendering neural materials on curved surfaces*. SIGGRAPH '22 Conference Proceedings.

Mohanty, A. (2020). *Multi layer Perceptron (MLP) Models on Real World Banking Data*. Becoming Human: Artificial Intelligence Magazine.

Rainer, G., Ghosh, A., Jakob, W., & Weyrich, J. (2020). *Unified Neural Encoding of BTFs*. In Computer Graphics Forum (Proceedings of Eurographics) 39(2).

Williams, L. (1983). *Pyramidal parametrics*. SIGGRAPH Comput. Graph. 17, 3 (July 1983), 1–11.