UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENT DE MATEMÀTICA APLICADA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Applied Mathematics

In-the-wild Material Appearance Editing using Perceptual Attributes

Master's Thesis

Master's Degree in Mathematical Research

AUTHOR: Subías Sarrato, José Daniel

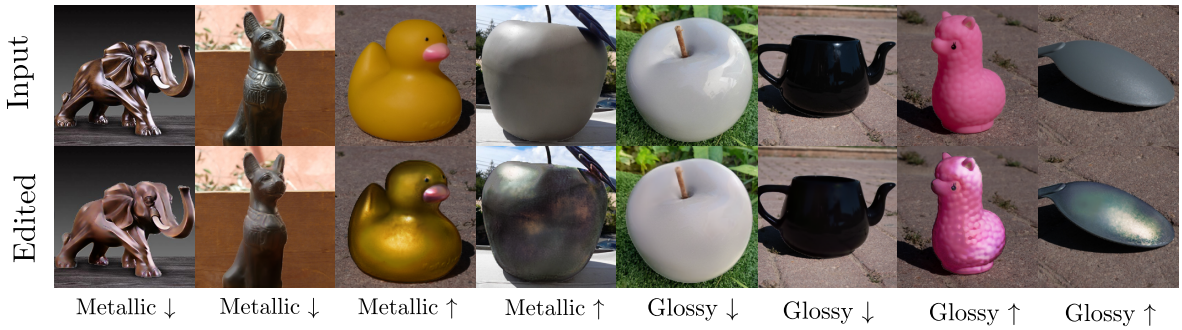
Tutor: Morillas Gómez, Samuel

External cotutor: LAGUNAS ARTO, MANUEL

ACADEMIC YEAR: 2021/2022

Master's Thesis

In-the-wild Material Appearance Editing using Perceptual Attributes



Author:

José Daniel Subías Sarrato

Supervisor:

Dr. Samuel Morillas Gómez, Universidad Politécnica de Valencia

Co-Supervisor:

Dr. Manuel Lagunas Arto, Amazon

Master's Degree in Mathematical Research
2021/22

Acknowledgments

I would like to thank my supervisors Samuel Morillas and Manuel Lagunas for the time they have invested in this project and Professor Diego Gutierrez from Graphics and Imaging Lab for trusting me and giving me the opportunity to solve such a complex problem.

I would also like to thank the IUMPA institute for the computational resources provided, without which the development of this project would not have been possible.

Finally, I would like to thank my family for their support throughout this year and my classmates for all their help in the master's courses and for making this year unforgettable.

Abstract

Intuitively editing the appearance of materials, just from a single image, is a challenging task given the complexity and ambiguity of the interactions between light and matter. This problem has been traditionally solved by estimating additional factors of the scene like geometry or illumination, thus solving an inverse rendering problem where the interaction of light and matter needs to be modelled. Instead, in this master's thesis, we present a single-image appearance editing framework that allows to intuitively modify the material appearance of an object by increasing or decreasing high-level perceptual attributes describing appearance (e.g., glossy or metallic). Our framework uses just an in-the-wild image as input, where geometry or illumination are not controlled.

We rely on generative neural networks and, in particular, on Selective Transfer Generative Adversarial Networks (STGAN) that allow to preserve high-frequency details from the input image in the edited one. To train our framework we combine pairs of synthetic images, rendered with physically-based ray tracing algorithms, and their corresponding ratings of the high-level attributes, given by humans through crowd-sourced user studies. Last, although trained on synthetic images, we demonstrate the applicability of our method on synthetic video sequences; and real-world photographs downloaded from online catalogs and manually taken using our mobile phones.

Index

Figure List

Table List

Chapter 1

Introduction

Humans are visual creatures, most of our input information comes just from sight. Indeed, visual data is a key aspect of how we perceive the world around us. We are capable of understanding the visual appearance of objects in a glimpse. However, although our visual system can understand it in a matter of milliseconds, such visual appearance is formed by a complex multidimensional physical interaction between light and matter that creates appearances of astonishing richness like pearls, northern lights, or the feathers of a bird. Painters and photographers have been throughout the years inspired by the beauty of those appearances, and more recently, digital artists and computer graphics professionals have tried to digitally synthesize them. Our society is increasingly relying on computer-generated imagery for daily tasks, with visual appearance and material appearance being a core aspect, not only for how we, humans, understand the world but also on how we communicate ideas, or on how we develop digital content.

We have learned to digitally simulate the physical interaction between light and matter for most appearances, however, how our visual system perceives them is not fully understood. Untangling the processes that happen in our visual system, and finding a direct relationship between our subjective impression and the physical parameters that govern the interaction between light and matter is an open challenge. Moreover, this problem is further aggravated since the mathematical models that simulate the interaction between light and matter are usually not intuitive nor predictable, highly dimensional, and only understood by professionals. In this master's thesis, we aim at alleviating this problem by combining novel machine learning methods, computer graphics, and human perception to develop a framework capable of offering an intuitive editing method for material appearance given just a single image.

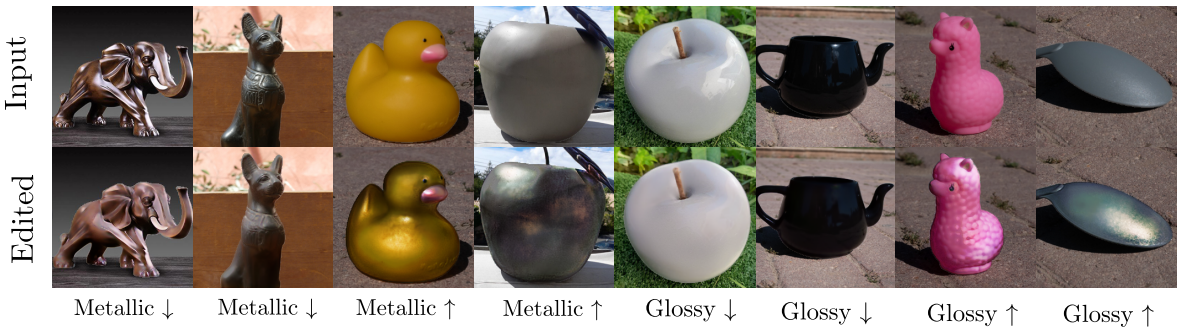


Figure 1.1: Top row: Given a certain image of a single object as input, our framework allows to edit the appearance of an object using high-level perceptual attributes. Bottom row: Our framework produces realistic edits for a variety of photographs of objects with different material appearance, illumination, and geometry. We can see how illumination conditions remain unchanged in the edited images even though they were not explicitly modeled in the framework. Arrows indicate a high (pointing up) or low (pointing down) value of the target high-level perceptual attribute.

We present an image-based framework that does not rely on any physically-based rendering but instead modifies directly the material appearance in the image pixels. It takes a certain image of an object as input and modifies the appearance based on varying the value of the desired high-level perceptual attribute describing appearance (see Figure 1.1). Recently, the popularization of generative deep learning models had allow us to design data-driven frameworks for image editing [3, 4, 5]. Since the image cues that drive the perception of such attributes can not be captured in a few image statistics, we rely on such generative neural networks to learn their relationship with material appearance and generate novel edited images.

To train our framework, a first approach might be collecting pairs of images the originals and the edited ones, where the edited examples were produced given a target high-level perceptual attribute value. This approach is not only tedious but also hinders the training process. Thus, we follow the work from Delanoy et al. [2], which is based on training a two-step generative framework using a wide dataset composed of a large variety of images, paired with high-level perceptual ratings obtained through user studies. However, this method needs additional geometrical information (as a normal map) of the target object as input as well. We thus devise an encoder-decoder architecture that allows us to send the relevant high-frequency information of the object's shape from the encoder to the decoder, removing the need for the normal map as the input.

To demonstrate the editing capabilities of our framework on a varied set of synthetic images and online photographs, we focus on two high-level perceptual attributes that are both common and easy to understand: metallic and glossy. We evaluate the consistency of our framework on a wide variety of scenes with different illuminations, materials, and geometries; and compare the results with the work from Delanoy et al. [2]. There, we observe that our method, although cheaper and with less input information, is capable to obtain more realistic edits of the input image. We also assess the temporal consistency by editing video sequences composed of frames rendered

using unseen illumination, geometry, and materials; and varying the camera viewpoint. There, we observe that our framework is capable of producing coherent outputs even when the additional temporal dimension is included.

1.1 Contributions

The contributions of this master thesis are the design, implementation, and evaluation of a generative-based model for single-image appearance editing. We propose a novel encoder-decoder network architecture and loss functions. With it, we are capable of achieving better reconstruction results than previous work [2] while using a computationally cheaper model and requiring only an in-the-wild single-image as the input (the previous work also requires additional geometry information). To summarize, the core contributions of this thesis are:

- **Model design**: Development of a new encoder-decoder architecture for image-base editing. Our model allows to introduce skip-connections that allow to send high-frequency information from the encoder to the decoder (Chapter 4).
- **Model training**: A methodology to train the proposed framework on image datasets that are paired with high-level human ratings of attributes describing appearance (Chapters 5 and 6).
- **Model evaluation**: Test the performance of the trained model through ablation studies and comparisons with the previous state-of-the-art method [2] (Chapter 7).

1.2 Outline

The outline of this master's thesis is as follows:

- **Chapter 2: Background**: A brief explanation of the fundamentals of computer graphics and deep learning to facilitate the comprehension of this master's thesis, since these are the main topics of this project.
- **Chapter 3: Related Work**: Introduction to the state of the art in appearance editing and description of previous work for single-image editing using geometrical information of the scene, which forms the baseline of this master's thesis.
- **Chapter 4: Our Framework**: Description of our proposed framework for single-image editing, without any additional information of the scene.
- **Chapter 5: Dataset**: Description of the datasets used to train and test our framework and how we collect this high-level perceptual data.
- **Chapter 6: Training Scheme**: Explanation of the training pipeline of our framework: loss functions, optimization algorithm, hardware specifications, and the input preprocessing.

– **Chapter 7: Evaluation and Results**: Ablation studies validate the training effects of our proposed framework with different datasets and the editing results of our best editing framework and its comparison with previous work.

– **Chapter 8: Conclusions**: Conclusions on the initial objectives of the project and proposals for future lines of work.

1.3 Academic Context

This project corresponds to my master's thesis for the master in Mathematical Research at the Polytechnic University of Valencia under the supervision of professor Samuel Morillas and co-supervised by Dr. Manuel Lagunas from Amazon. We also collaborated with the *Graphics and Imaging Lab*, a research group at the University of Zaragoza, which provides us with the dataset and the code necessary to develop this master's thesis. The group's work focuses on computer graphics, conducting research in areas of physically realistic rendering, image processing, computational photography, virtual reality, or applied perception, among others. Professor Diego Gutierrez has collaborated and discussed the approach and results of the framework described in this thesis.

1.4 Methodology

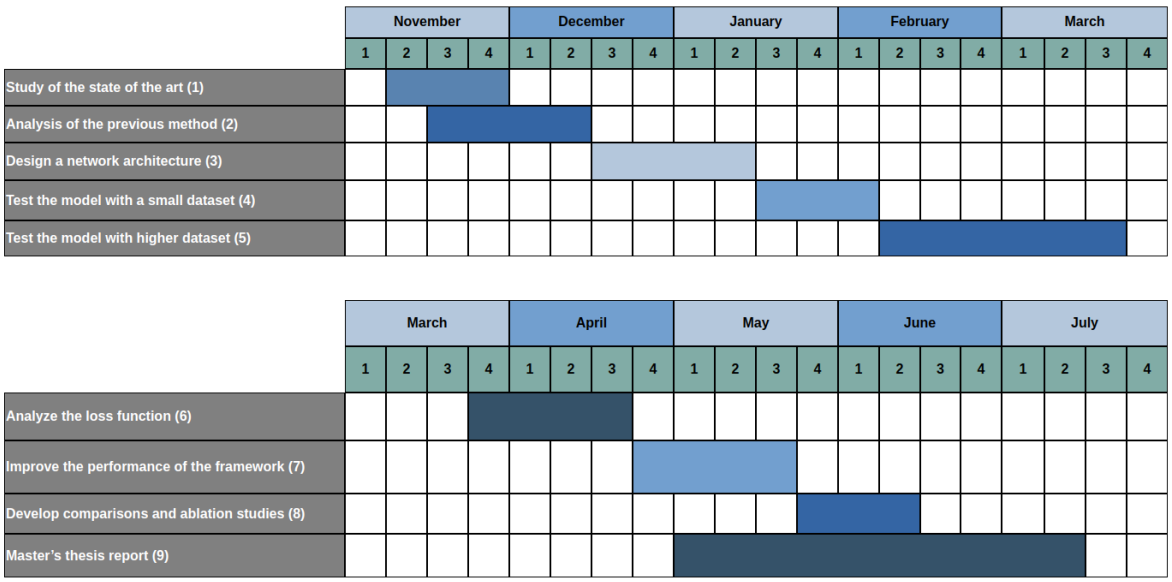


Figure 1.2: Gantt chart of the project schedule.

The timeline followed to achieve the different objectives of this master thesis is shown in the Gantt chart in Figure 1.2. The time dedicated to the project is distributed over nine months using the following methodology:

1. Study of the state of the art in single-image appearance editing.
2. Analysis of the previous state-of-the-art method from Delanoy et al. [2].
3. Design a novel network architecture for appearance editing. In particular, compared to previous work, our network does not require additional geometry information in the input. Indeed, it just needs an in-the-wild image that could be casually taken with our phones.
4. Test the model with low-resolution images and a small dataset.
5. Use high-resolution images and a higher dataset.
6. Analyze the noise in the weights of the loss function and adapt it to the new dataset.
7. Improve the performance by modifying the network and training parameters.
8. Develop thorough comparisons by doing ablation studies and obtaining the performance of the previous state-of-the-art method from Delanoy et al. [2].

1.5 Tools

We take as baseline the Python code from the work *A Generative Framework for Image-based Editing of Material Appearance using Perceptual Attributes* [2] implemented in PyTorch [6]. The datasets used for training and evaluation of the framework were processed with the OpenCV [7] and torchvision libraries. The framework was trained with a Nvidia GeForce RTX 3090 GPU with an integrated memory of 24 GB. GitHub was used as the version control tool.

Chapter 2

Background

This section gives a brief introduction to both computer graphics and deep learning. Section 2.1 explains the main concepts of computer graphics, starting with the description of a 3D scene, then, we explain how to model the interaction of light and matter through the light transport equation, and last, we describe how the scene parameters and the light transport equation can be used to generate photorealistic images. Section 2.2 explains the basis of deep learning, introducing how these kinds of algorithms are classified depending on the way of training; then, we introduce the most common deep learning models used nowadays, especially for image processing; and last we describe the phases that compose of the training process of deep learning models.

2.1 Computer Graphics

In computer science, computer graphics is the field that gathers all the algorithms and software dedicated to manipulating and creating images on a computer. It is an ubiquitous field, we are interacting with computer-generated images daily: in video games or movies, for architectural visualizations or in the medical industry. The usual workflow to generate a synthetic image starts with the design of a 3D scene, setting its parameters, and then ruining an algorithm capable of simulating the interaction of light with such scene and generating a photorealistic image.

2.1.1 Scene Parameters

A scene is the set of components that makes up the 3D synthetic place that we want to display on the computer screen. A scene contains the following elements:

- **Geometry**: The set of shapes that represent the 3D objects in the scene, i.e spheres, cubes, triangles, or any other object.
- **Camera**: The point of view of the observer. This sensor transforms the light, that reaches it, into a 2D image of the observed region of the scene.
- **Light sources**: The set of light-emitting objects like bulbs, the sun or other real objects that emit light by themselves.

– **Materials**: They model the final appearance of each object located in the scene (e.g plastic, metallic, wood or cloth). Materials define the interaction between light and objects' surfaces. As we will explain below, all materials are characterized by a light distribution function.

Figure 2.1 shows a simple 3D scene. There, we can see the geometry of a sphere with an orange-like material. Such sphere is being lit by a light bulb. The scene is being observed by the camera which captures the visual appearance of the sphere.

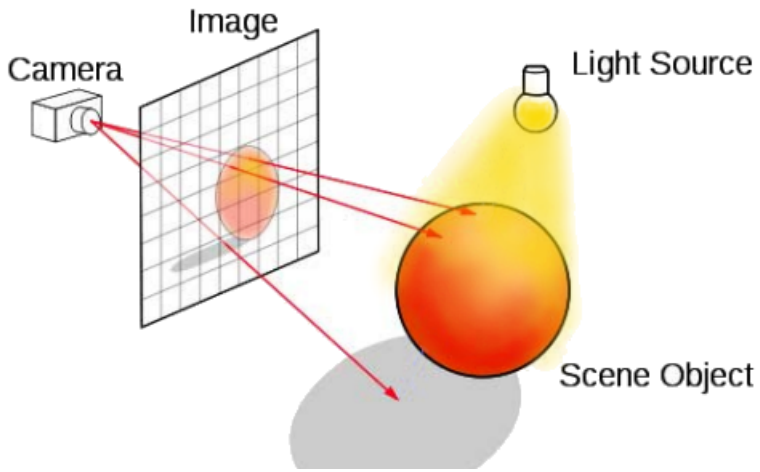


Figure 2.1: Scene composed of an orange sphere illuminated by a single light source. The image generated by the camera captures the orange-like appearance of the sphere, in addition to the shadows created by the interaction between light emitted from the bulb and the object. Image taken from [8].

Once we have set the parameters of the 3D scene, it is necessary to transform it into a 2D image. To do so, we use algorithms that simulate the interaction between light and matter. Those algorithms are called rendering algorithms and rely on light transport models that, as we will see below, characterize the interaction between light and matter.

2.1.2 Light Transport

Our visual system assigns colors to visible objects since light rays bounce on them and reach our eyes. When light impacts an object, the rays may be reflected, refracted or transmitted. The incoming light ray which travels with direction w_i from a light source with position c, and impacts at a point x at a surface, is given by:

$$L_t\left(x, w_i, c\right) = \frac{p}{|c - x|^2}. \tag{2.1}$$

Where p is the light power of the light source. Then the outgoing light with direction w_o that reaches our eyes or a camera (in a 3D scene) is a percentage of the incoming light, and it is strongly related to the material properties of the objects. The design of rendering algorithms involves defining mathematical models to relate all the scene

parameters. These models are called light transport models and describe the interaction between light and matter before light rays reach the camera.

The Render Equation

The render equation [9] is the usual mathematical model to estimate the outgoing radiance at a point x in a surface depending on the scene parameters, as is shown in Figure 2.2. This equation is defined as:

$$L\left(x, w_o\right) = \int_{\Omega} L_i\left(x, w_i\right) f_r\left(x, w_i, w_o\right) \left|n \cdot w_i\right| dw_i. \tag{2.2}$$

Where L is the outgoing light with direction w_o from all rays w_i, which impacts the object's surface at the point x over the hemisphere Ω. L_i is the light power of the i^{th} ray and usually is given by Equation 2.1. The function f_r is the Bidirectional Radiance Distribution Function (BRDF) and defines the percentage of incoming radiance that leaves with direction w_o, playing an important role in the object's appearance. The term $|n \cdot w_i|$ denotes the angle between the incoming direction and the normal surface n. The light power grows as the angle decreases. The angle falls within the interval $[90, 0)$ concerning the surface normal.

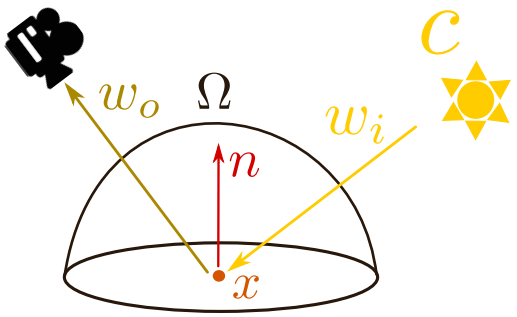


Figure 2.2: Light-matter interaction, where w_i is the direction of the incoming light from the light source c and w_o the outgoing direction of the reflected light. Ω express the integration hemisphere with center x.

Replicating the real behavior of the light is a complex task since the light rays interact with all objects throughout the scene. Therefore, when we rely on the Equation 2.2 at a point x, some rays do not necessarily have to come from a light source, they may interact with another object before.

Local Illumination

The render equation gives us the light that leaves from the object's surface with a specific outgoing direction. The light radiated from emitting objects travels throughout the scene interacting locally at the objects' surfaces. Local illumination considers only light from the original visible-light sources at the surface, rejecting the reflected light from other objects, as is shown in Figure 2.3. Therefore, in that case, Equation 2.2 may be approached easily as the following summation:

$$L_d(x, w_o) \approx \sum_{i \in \mathcal{I}} L_i(x, w_i, c_i) f_r(x, w_i, w_o) |n \cdot w_i| . \tag{2.3}$$

Where $\mathcal{I}$ is the set of indexes of visible lights from x. Nevertheless, this is an incomplete approximation of how light would behave in the scene in reality, so complex visual effects are not modeled.

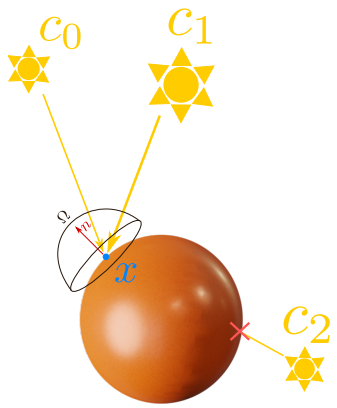


Figure 2.3: Example of a scene with three lights and an orange sphere. The lights c_0 and c_1 contributes to compute the outgoing light at the point x form direct illumination of light. The light source c_3 is occluded by the sphere and it does not contribute to the light estimation at x.

Global Illumination

Assuming that light travels directly from the light sources without interacting with other objects, is an inaccurate simulation of its real behavior. Global illumination differs from local illumination because it also calculates light as it would travel throughout the entire scene. This involves taking into account reflected light from other objects present in the scene together with direct illumination.

Let's say we have a single ray emitted from a light source, assuming that it travels straight ahead from the light source, when it impacts an object its trajectory changes and may reach another object as is shown in Figure 2.4. Therefore Equation 2.2 must be computed recursively at each bounce.

Computing the global illumination enhances the realism of the final image because of tries to approach the real behavior of the light accurately. This leads to life-like aspects of the objects' surfaces, and mathematically the outgoing light that finally reaches the camera is expressed as is shown in Equation 2.4, where L_g represents the global illumination, L the indirect illumination and L_d the direct illumination,

$$L_g(x, w_o) = L(x, w_o) + L_d(x, w_o) . \tag{2.4}$$

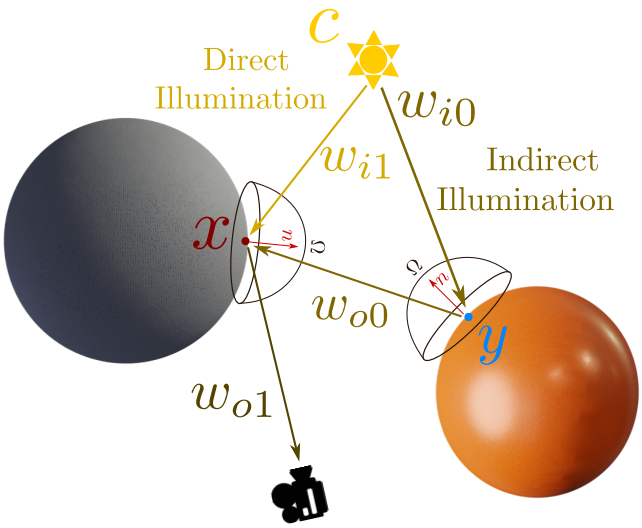


Figure 2.4: Example of a bounced ray which travels from a light source c. The render equation is computed in the orange-like sphere at point y, to determinate the reflected light power that reach the second gray sphere. The incoming radiance at x is the sum of both indirect and direct light.

Materials: Bidirectional Radiance Distribution Functions

In the real world, we can find different material appearances whose astonishing visual appearance is very hard to mathematically replicate (e.g pearls, bird's feathers, etc). As we commented above, the outgoing percentage of light is related to the material properties of an object. Bidirectional Radiance Distribution Functions (BRDFs) may be classified among parametric, depending on parameters like the wavelength of light ; and non-parametric, derived from real-data measurements. Instruments like gonioreflectometers allow us to capture the outgoing energy of light rays tossed from a light source to a slice of certain material. For simplicity, we explain here a few of the most common materials and refer the interested reader to [10, 11, 12, 13, 14]. Montes et al. [15] gives a comprehensive overview about the different kinds of BRDF mathematical models.

A BRDF is a function of real variables that defines light reflected at an opaque surface. Figure 2.5 graphically shows examples of the most common ones, which define the light interaction for the following materials:

- **Diffuse**: These materials distribute light uniformly throughout Ω. Therefore from any point of view, the outgoing radiance remains constant. The surface only has a single color and its intensity and shadows vary depending on the distance to the light sources.
- **Specular**: The outgoing radiance changes around the point of view. Materials like mirrors are mostly specular, so the angle θ_o between the reflected ray and the normal is the same as the angle θ_i between the normal and the incoming ray.

– **Glossy**: This light iteration is a special case of specular reflection. Light is distributed around the reflected ray w_r from different angles than the incident one. This produces smooth reflections at the surface as is illustrated in Figure 2.5.

– **Metallic**: These materials are another variant of the specular BRDF. Usually, perfect specular BRDF assumes perfect-smooth surfaces, then objects reflect the surrounded environment. However, for metals that statement is not necessarily true since some of them (e.g gold aluminum or bronze) changes the color of their specular reflection. This is due to the roughness of the object's surface changing the angle of the outgoing direction.

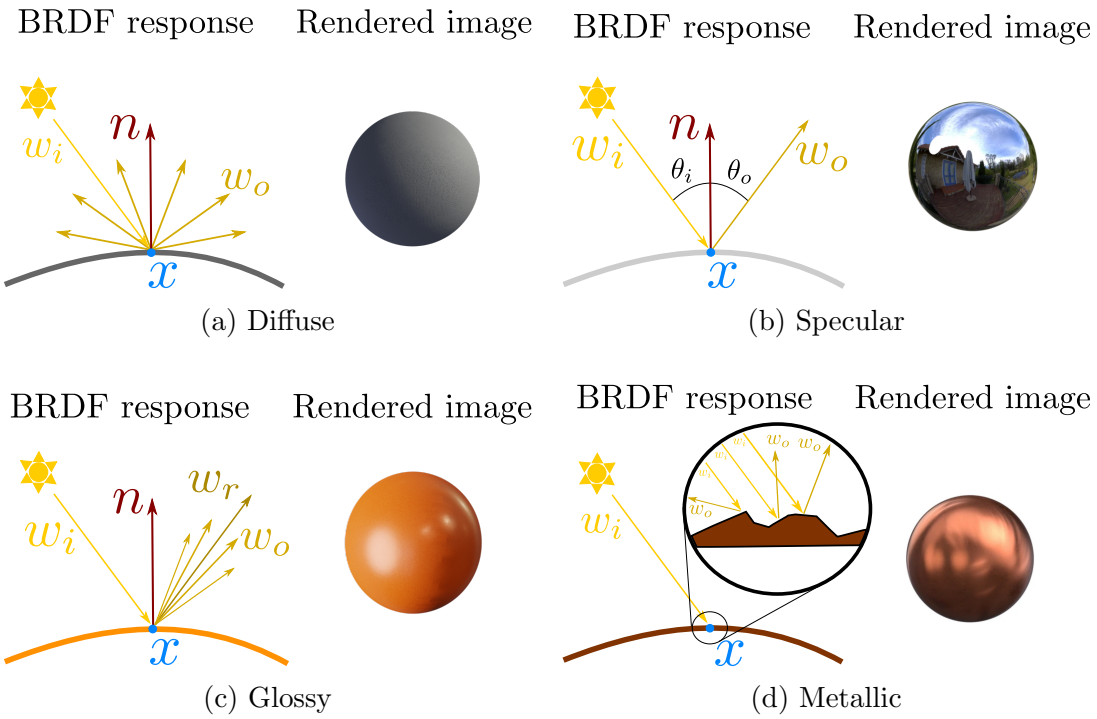


Figure 2.5: Upper-left: BRDF response of a pure diffuse surface and a rendered gray sphere with such BRDF without any specular reflection at the surface. Upper-right: BRDF behavior of a perfectly specular surface, and a rendered image with this BRDF where we can see the environment around the sphere perfectly reflected. Lower-Left: BRDF of the interaction of glossy materials and a rendered orange ball with smooth reflections at the surface. Lower-right: Light interaction at non-smooth surface and a rendered bronze sphere which has reddish color and does not reflect the surrounded environment.

Not all materials that we interpret are pure diffuse, specular, glossy or metallic, there may be a mix of them or other interactions. For example, plastics have specular and diffuse parts without glossy reflections, dielectric materials like glass also refract the incident rays, or skin has additional effects like sub-surface scattering.

2.1.3 Image Rendering

Image rendering is the process of generating realistic images once we have set the parameters of the scene. Above we defined the interaction between light, shapes and materials, through light transport models and the render equation. Now we need to understand how to capture the light energy into the pixels that will conform the rendered photorealistic image of the original 3D scene.

Formally, the camera model is defined by the up, left and front vectors, which compose an orthogonal base on $\mathbb{R}^3$ with origin o as is shown in Figure 2.6. In addition, a view plane with a grid-like topology is defined to represent the final image. Each cell is a pixel to store the incoming light in its region of the plane, so the image's resolution depends on the number of pixels.

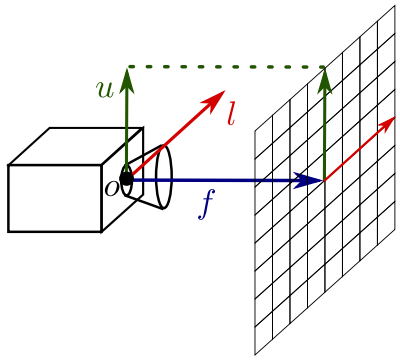


Figure 2.6: Representation of a simple camera where u, l and f are the up, left and front vectors respectively. The high and width of the view plane depends on the norm of u and l. The point o expresses the viewpoint of the scene from the camera.

The goal of rendering algorithms is to find for each pixel the object that is seen at that pixel's position in the image. Each pixel looks in a direction and any object seen by a pixel must intersect the viewing ray, a line traced from the viewpoint in the direction that pixel is looking. The object we want is the one that intersects the view ray nearest to the camera since it occludes any other objects behind it. Once the view ray intersects the object's surface, then Equation 2.4 is computed to estimate the pixel's color, as Figure 2.7 illustrates. When view rays do not intersect with anything, their pixels have to take the background color.

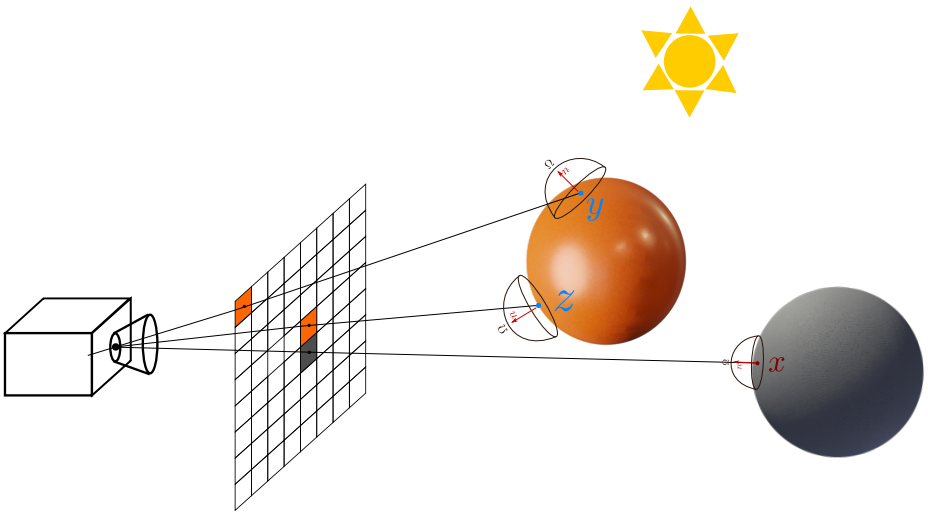


Figure 2.7: Schema of the ray tracing algorithm. The view rays emitted from the camera determinate the nearest object seen by each pixels and the intersection point at its surface. The orange-glossy sphere occludes the upper part of the gray-diffuse, so those pixels take an orange color.

Usually, view rays take direction d from the camera center o to pixel's center p_c, in such way that $w_o = -d$ from the point of view at the intersected surface. Formally, the direction + of a view ray is defined as the following vector:

$$d = p_c - o. \tag{2.5}$$

To enhance realism, some view rays per pixel may be traced to sample better the objects' surfaces, so the color of a pixel is the average of all of them. Once the image has been rendered, the pixel's colors may have a High Dynamic Range (HDR), so that, their values exceed the color range of the computer screen. Then, a tone mapping operator to translate pixel values to a Low Dynamic Range (LDR) must be applied. An example of this may be clamping the pixels' colors by a threshold or normalize all of them by the maximum value of the rendered image. The common choice is the gamma correction defined as:

$$p_{LDR} = E \cdot p_{HDR}^{1/\gamma}. \tag{2.6}$$

Where p_{LDR}, E and p_{HDR} are the final value of a certain pixel p in LDR, the exposure constant (commonly set to 1) and the original value of a certain pixel p in HDR respectively. Finally, *gamma* is the selected value for changing the brightness of the input image. Usually the images are represented through the RGB color space, that is, they have one channel for each intensity of red (R), green (G), and blue (B) colors.

2.2 Deep Learning

In this section we begin by defining the different types of deep learning training algorithms. Next, we describe the most common deep learning models used for image processing and, finally, we describe in detail the training process of such algorithms. Artificial Neural Networks (ANNs) are algorithms, which are inspired by the biological behavior of the human brain. Similar to our brains having neurons interconnected to each other, ANNs have neurons that are grouped in layers, which are linked to each other. The neurons take as input some data, apply a transformation and propagate the output to others. Deep learning is a subset of machine learning methods focused on the design and training of ANNs with dozens of layers (which why is called deep learning) to solve some specific task. These algorithms have a great impact nowadays in tasks such as classification problems, self-driving cars, medical diagnostics and demand prediction.

2.2.1 Types of Training

When training a deep learning model a large number of relevant examples are required to allow the model to learn correctly from them. ANNs have a set of trainable parameters optimized throughout training to perform a specific task from data. The set of examples is called the training dataset and it is composed of labeled or unlabeled data, so deep learning algorithms can be classified depending on the way of training into the following groups:

- **Supervised**: If the model is training via supervised learning, then examples need to be labeled by a ground truth since this information is what the ANNs must learn from data. The trainable parameters are self-optimized through the training process to generate an output as similar as possible to the ground truth. Therefore supervised methods seek to minimize a loss function, which measures the difference between the outputs and the ground truth.

- **Unsupervised**: The algorithms are trained with unlabeled data to solve a classification or association problem. In contrast to supervised training, unsupervised methods exhibit to find similarities or discrepancies between data in the training set. Therefore models group data into some clusters. Usually, we should specify the number of clusters before training the algorithm.

- **Adversarial**: An ANN called generator should generate outputs to trick a second ANN called discriminator playing an adversarial game. That is, they compete with each other until a desirable equilibrium is reached. The generator takes some data and tries to generate an output similar to the training data. The discriminator tries to discriminate between the generated data and the real ones.

The dataset is usually divided into three different splits: training, validation and test splits, which represent the 70 %, 15 % and 15 % of the whole dataset respectively. Often the input data are processed to facilitate the model's learning. This process is commonly called data cleaning and consists of detecting outliers and dropping them.

2.2.2 Different Model Architectures

The architecture of an ANN is the set of layers and their connections that transform the input data to generate the predicted output. Above we described the different ways of training a deep learning model, which are characterized by the training dataset. Now we will introduce the most common ANN architectures to extract information from image datasets and find patterns in them.

Convolutional Neural Networks

In deep learning, a Convolutional Neural Network (CNN) is a class of ANN, most commonly applied to image processing. CNNs have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

Those algorithms are a special family of ANNs that contain convolutional layers to extract features and details from the input images by 2D tensors of weights usually called kernels. Before going further, we should briefly review what are convolutions and kernels. Formally, the convolution operator $*$ on two functions f and g produces a third function $(f * g)$ that expresses how the shape of one is modified by the other. This operator is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau. \tag{2.7}$$

Where τ is the shift of the function g. However, since the algorithms run into a computer we approximate the Equation 2.7 by a summation defined as:

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau). \tag{2.8}$$

Using the terminology of convolutional networks, the function f is referred as the input image $\mathbf{I}$ and g is the kernel $\mathbf{K}$. Since the kernel slides across the height and width of the image, which is expressed as a tensor, then the 2D discrete convolution is defined as:

$$f(i,j) * g(i,j) = \sum_{n} \sum_{m} f(n,m) * g(n-i, m-j). \tag{2.9}$$

Where n and m are the numbers of rows and columns respectively. Figure 2.8 shows an example of the 2D convolution operator over a square tensor.

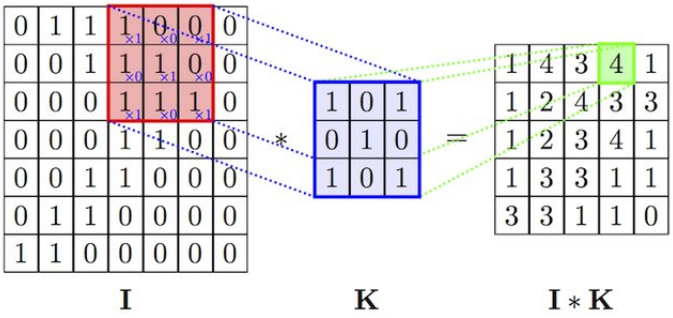


Figure 2.8: Example of a 2D-convolution over a square tensor $\mathbf{I}$. A patch with size of 3×3 px. in the input tensor $\mathbf{I}$ is multiplied one by one by the weights of the kernel $\mathbf{K}$ with size of 3×3, and then all of them are summed together into the destination tensor. Image taken from [16].

The size of the output image can be changed by setting a stride parameter, which controls how the kernel slides across the input tensor, as is shown in Figure 2.9. The variables s_v and s_h represent the vertical and horizontal strides respectively. If the kernel is situated at the pixel $\mathbf{I}[i, j]$ of the input, the following pixel is $\mathbf{I}[i + s_h, j]$. Since the kernel crosses the image from left to right starting at column t, once the right-vertical edge is reached by the kernel, it starts again from the left-vertical edge at pixel $\mathbf{I}[t, j + s_v]$.

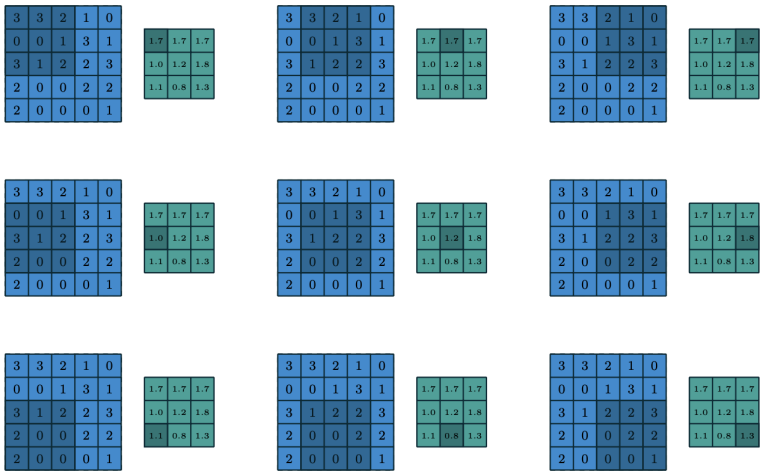


Figure 2.9: Example of a kernel (green tensor) with size 3×3, which slides over a image (blue tensor) with size of 5×5 px. It starts at pixel $\mathbf{I}[1, 1]$ and crosses the image from left to right with $s_h = 1$ and $s_v = 1$, therefore the output image has a size of 3×3 px. Image taken from [17].

Depending on the starting pixel and the stride, some weights may be placed out of the input image while the kernel is processing it. A usual practice called padding consists of adding a constant border (usually initialized to 0) surrounding the image, Figure 2.10 illustrates an example of technique, where p defines the thickness of the constant area around the image.

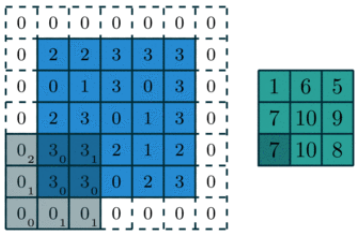


Figure 2.10: A image with size of 5×5 px; the stride causes that the 5 weights of the kernel, with size 3×3, to be left out the images. Therefore the image is bounded by a region of zeros with thickness $p = 1$. The tiny numbers at the lower-right corner of the gray square are the kernel's weights. Image taken from [18].

Given a certain single-channel image with size of $n \times m$ px; and a square kernel of size k, then the height h and width w of the output image are defined as:

$$w = \left\lfloor \frac{n + 2p - k}{s_v} + 1 \right\rfloor, \tag{2.10}$$

$$h = \left\lfloor \frac{m + 2p - k}{s_h} + 1 \right\rfloor. \tag{2.11}$$

Convolutional layers are composed of filters, which are sets of kernels. As output, they provide feature maps that indicate the locations and strengths of detected features in the input image. So the trainable parameters are the filters, optimized to find the most relevant details in the input image. Let's say we have a certain image $\mathbf{I}$ with size $n \times m \times c$, where c is the number of channels on it. Then the image is passed through a single layer composed of $\mathbf{F}_i$ filters with as many k-squared kernels as input channels. Each filter will produce an image $\mathbf{O}_i$ defined as:

$$\mathbf{O}_i = \sum_{\mathbf{K}_j \in \mathbf{F}_i} \mathbf{K}_j * \mathbf{I}_j. \tag{2.12}$$

Where $\mathbf{I}_j$ denotes the j^{th} channel of the image. Thus, as is illustrated in Figure 2.11, the layer's output is a feature map with $\mathbf{O}_i$ images with size of $w \times h$ px. following Equation 2.11. After that, some activation function may be applied to each value of the feature map to introduce non-linearity in the model. A common choice is the ReLu function, defined as:

$$ReLu(\mathbf{z}) = \max(0, \mathbf{z}). \tag{2.13}$$

Where $\mathbf{z}$ is some image from the feature map, so $\mathbf{z} \in \mathbb{R}^{w \times h}$.

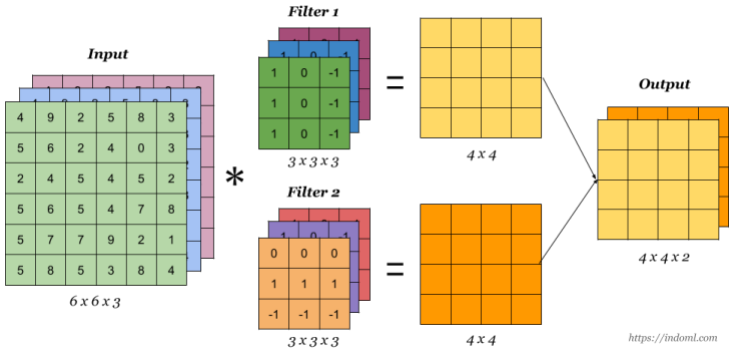


Figure 2.11: Example of a convolutional layer. A RGB image is taken as input. The layer has two filters composed of three square kernels (one per input channel) with size 3×3, that start to slide through the matrix at pixel $\mathbf{I}[1, 1]$ with $s_v = 1$ and $s_h = 1$. The output feature map is composed of the sum of the convolved channels separately, from both filters. Image taken from [19].

Convolutional Encoder-Decoder Architecture

Previous section explains how CNNs extract information from images. Now we introduce a network architecture that evolved from the traditional CNNs. The main idea is to compress the input image into a low-dimensional representation and then it is decompressed to assemble a precise output. The compression phase is carried out by a CNNs called encoder while its output is reconstructed by another CNNs known as the decoder.

Before continuing, we should briefly review what are max pooling and up-sampling operations. First one is typically added to CNNs following individual convolutional layers. Max pooling reduces the dimensionality of images by modifying the number of pixels in the output from the previous convolutional layer to speed up calculations. This consists of sliding a square kernel with size $k \times k$ using and strides $s_h = k$ and $s_v = k$, but now the output is the maximum between image values inside the kernel as is shown in Figure 2.12.

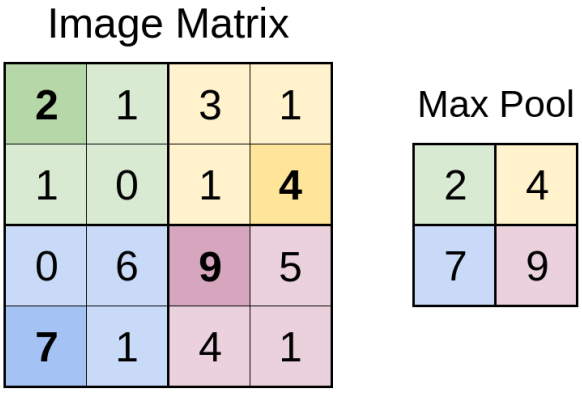


Figure 2.12: An example of max pooling over a image with size 4. The kernel slides across the matrix with $s_h = 2$ and $s_v = 2$ and selects the maximum at each pixel patch. Therefore the max-pooled image is as matrix with size 2. Image taken from [20].

Max pooling layers are included in the encoder architecture to compress the input image without losing relevant information. As we mentioned above, the decoder should decompress incoming information from the encoder, so we need to include up-sampling layers in its architecture. The up-sampling operator has the opposite effect as max pooling, the image's dimensions are multiplied by a scale factor s_c. Let's say we have an image with size of $n \times m$ px; then it is re-scaled to $h \times w$ px. following the relations:

$$w = \lfloor n \times s_c \rfloor, \tag{2.14}$$
$$h = \lfloor m \times s_c \rfloor. \tag{2.15}$$

To determine the pixels' values in the output image an interpolation algorithm (e.g nearest neighbour, linear, bilinear, trilinear, bicubic, ...) is computed. We refer curious readers to [21, 22, 23] for more detailed information.

U-Net [1] is the most common convolutional encoder-decoder architecture, it was developed for biomedical image segmentation. The encoder applies convolutions each followed by a ReLu and a max pooling operation to reduce the spatial information while feature information is increased. On the other hand, the decoder combines the feature and spatial information, through a sequence of up-sampling layers to reconstruct the final image. The deepest feature map produced by the encoder is usually called latent code, which is a low-dimensional representation of the input.

To enhance the output quality, high-resolution information is sent from the encoder to the decoder via skip connections, that is, feature maps of the encoder's layers are concatenated to the decoder's feature maps as is shown in Figure 2.13. Also, the pixels of each channel in the final image are translated to the interval $[0, 1]$ by the *softmax* function, defined as:

$$\sigma(\mathbf{z}_{ic}) = \frac{e^{\mathbf{z}_{ic}}}{\sum_{j=1}^{K} e^{\mathbf{z}_{jc}}} \quad for\ i = 1, 2, \ldots, K. \tag{2.16}$$

Where $\mathbf{z}_i$ is the value of a pixel of a channel c formed of K pixels.

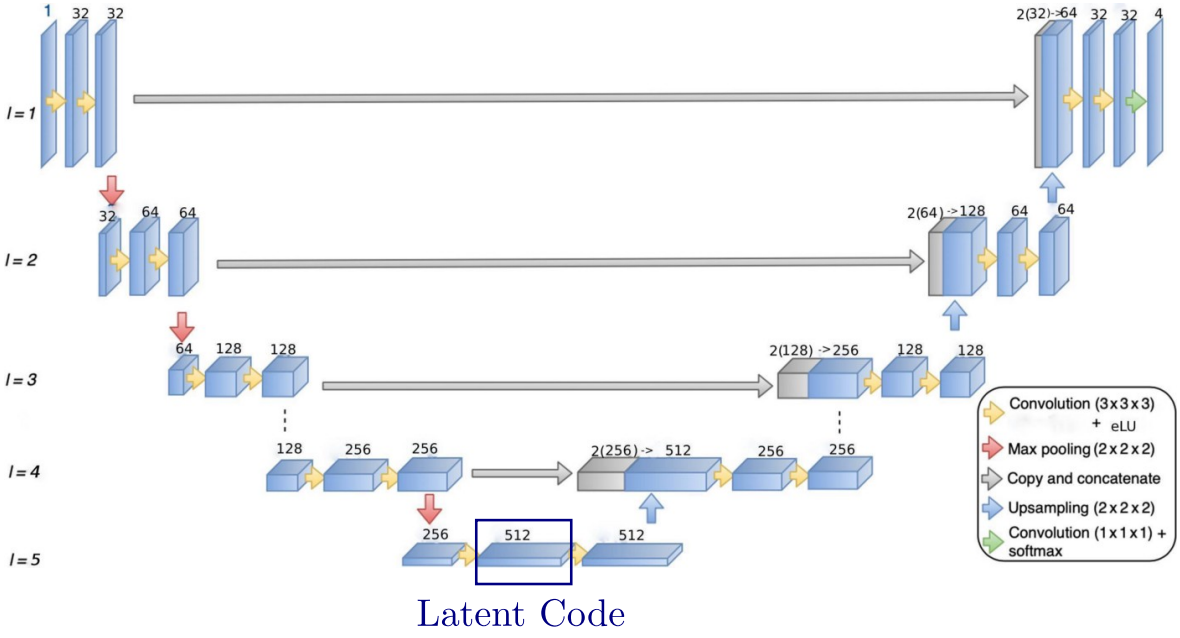


Figure 2.13: Schema of a U-Net architecture [1], both encoder and decoder are composed of five convolutional layers. Yellow arrows indicate a convolution operation followed by a ReLu function. Blue blocks are the feature maps of each layer, and black numbers above them denote their number of features. The latent code is decompressed by up-sample layers, blue arrows, assembling an RGBA image (one channel for red, green, and blue intensities and another opacity channel). Skip connections between encoding and the decoding layers are represented by gray arrows while the gray blocks express the concatenated feature map. Green arrow denotes the *softmax* function applied to the final image. Image taken from [24].

2.2.3 Generative Adversarial Networks

Generative adversarial networks (GANs) are algorithmic architectures that use two ANNs, pitting one against the other, to generate new synthetic instances of data that can pass for real data. GANs' potential is huge because they can learn to mimic any distribution of data. For example, GANs can be taught to create photorealistic images indistinguishable from our perception.

To understand GANs, we should know how generative algorithms work, and for that, contrasting them with discriminative algorithms is instructive. Discriminative algorithms try to classify input data, that is, given the features of an instance of data they predict a label or category to which that data belongs. For example, given a certain image of a pet, a discriminative algorithm could predict whether the image is a cat or a dog. When this problem is expressed mathematically, the image is defined as x and its class is demoted by y. Therefore the discriminator's output is the following probability:

$$D(x) = p(y|x). \tag{2.17}$$

On the other hand, generative algorithms do the opposite. They attempt to predict

features given a certain label. Therefore these algorithms try to learn the relation between x and y to generate data as alike as the real ones.

As we mentioned above, the training process is a two-contenders game. The ANN called the generator G, produces new data instances, while the other, the discriminator D, evaluates them for authenticity; i.e. the discriminator decides whether each instance of data that it reviews belongs to the actual training dataset or if it has been created by the generator. In the formal sense, D attempts to minimize the difference between their predictions and the real labels. In terms of probability, minimizing this error equals maximizing the cross entropy between labels and the discriminator's outputs. The cross entropy function H between two probability distributions p and q is defined as:

$$H(p, q) = -\mathbb{E}_{x\ p(x)} \left[-\log q(x) \right]. \tag{2.18}$$

Being x a single training data. Since in classification tasks, the random variable is discrete. Hence, the expectation $\mathbb{E}$ can be expressed as the following summation:

$$H(p, q) = -\sum_{x \in X} p(x) \log(q(y)). \tag{2.19}$$

Where X denotes the whole training dataset. Therefore D tries to maximize the following function:

$$\max_D V(G, D) = \mathbb{E}_{x\ p_{data}(x)} \left[\log D(x) \right] + \mathbb{E}_{z\ p_z(z)} \left[1 - \log D(G(z)) \right]. \tag{2.20}$$

On the other hand G wants to maximize the similarity between its generated data and the real ones to trick D. So the generator's loss function is given by:

$$\min_G V(G, D) = \mathbb{E}_{z\ p_z(z)} \left[1 - \log D(G(z)) \right]. \tag{2.21}$$

The ideal equilibrium point for both functions is reached when the generator models the real data correctly and the discriminator returns a probability of 0.5 for the fake data, meaning these are equally distributed as real data.

Usually, both the generator and the discriminator are trained concurrently from scratch and their behaviors should be stable through the training. Otherwise, the GAN may enter in what is called as mode called, that is, the generator learns to trick the discriminator, but with low-variability data. For example, a face generator should produce a different face for every distinct input. However, if a generator produces an especially plausible output, the generator may learn to produce only that output.

Image-to-image translation problems are an active field of research due to their application in video and image editions for films, video games and social networks industries. When working with images, AttGAN [4] is a regular model to address those problems. This architecture is composed of a convolutional encoder-decoder

image generator, that edits a set of features in the input image adding information in the latent code without manipulating other aspects of the image. The discriminator attempts to detect the edited features and discriminate between real and fake images.

2.2.4 How to Train a Deep Learning Model

Once the most common architectures for image processing have been described, we move on to describe the training process of an ANN from scratch. Following sections explain the remarkable phases of training deep learning models to perform a specific task once we have collected a large-varied training dataset.

Data Augmentation

The dataset should be varied, otherwise, it could result in an insufficient adjustment of the model to the task to be performed (i.e., underfitting) or memorization of the dataset. The ANNs' ability to generalize is strongly influenced by the variety of training data.

Data augmentation is a technique used to increase the amount of training data by adding slightly modified copies of already existing data or newly created synthetic examples. It helps to avoid overfitting, that is, the ANN failure in the learned task when processing unseen data. For example, when working with images, if the training dataset is not varied enough, an augmented version of it with rotation, mirroring or color shifts may help the model to generalize properly. Also, statistical techniques like normalization or standardization to avoid bias to large numbers may be applied as well.

Optimization

To train the model, each sample on the training set or a group of them, called batch, is passed through the network to produce an output. Then this output is used to compute a metric (e.g L_1 or L_2 norms) or a loss function (e.g. Mean Square Error) that assesses the capacity for the training task. Once the error has been calculated then it is back-propagated computing the gradients of the loss function concerning the trainable parameters.

Optimization algorithms are used to reduce the gradients by adjusting the trainable parameters' values. These algorithms (e.g Stochastic Gradient Descent, Adam, Adadelta) have a parameter called learning rate which is multiplied by the values of the gradients and allows us to control the convergence speed. This process is called the back propagation algorithm [25] and is repeated iteratively for each batch or sample.

The number of iterations to process the whole dataset is called an epoch. The optimization process usually terminated when the loss function obtained with the training data is no longer minimized with the optimizer. That fact indicates that the optimizer reaches a local or global minimum.

Validation

After several epochs the validation dataset is passed through to the model, to compute the validation loss function. This shows us feedback about how well the model is learning the specific task allowing us to detect overfitting. For example, a significant raising up of the validation through continuous epochs indicates that the model suffers from overfitting to the training data.

Model Evaluation

Once the model has been trained we must evaluate it using the test dataset that contains unseen data through the learning process. The test set is therefore a set of examples used only to assess the performance (i.e. generalization) using a loss function. This allows us to provide an unbiased evaluation of a final model fit on the training dataset. The test dataset also lets us compare the performance of diverse models to carry out the same task.

When working with generative models, there are no examples to make a comparison between the generator's outputs. For example on image-generation problems, the output images have never been taken. Thus, to evaluate the performance of the final model we can rely on perceptual judgments with user studies, where groups of non-expert people assign a rate to each synthetic generated and real image.

Chapter 3

Related Work

This chapter introduces the state of the art in appearance editing. Section 3.1 gives a review of the literature that studies how our perceptual system perceive the materials that are present in the real world. Then Section 3.2 introduces most common datasets that collect computational models of material from real-world appearances. After that, Section 3.3 focuses on describing the different methods to edit the image cues that drive the perception of the materials. Finally, Section 3.4 explains the most recent framework for single-image editing, which is the baseline for this master's thesis.

3.1 Visual Perception

Studying how our visual system interprets our world is a longstanding goal in computer graphics or applied optics among others; and is yet to be understood [26, 27]. Our perception of an object is strongly related with its material properties, but also involves some external factors such as geometry [28, 29], illumination [30, 31, 32] as well as motion [33, 34]. Some works try to find the stimulus that we use to infer appearance properties such as glossiness [32, 35, 36], translucency [37, 38, 39] or softness [40, 41] separately. Other works try to understand the perceptual cues used by artists when depicting materials in realistic paintings [42, 43] or determine the most relevant colors' properties that capture our attention when we observe a painting [44, 45]. Recent works [46, 47] point to human perception being guided by non-linear statistics, the same as complex deep learning models. Thus some appearance editing approaches rely on deep learning techniques to manipulate the image cues which guide our perception [2, 48] or measure the similarity between materials [49, 50]. Our framework is a deep learning model that learns the image cues that drive our visual perception to properly edit the appearance of materials in photographs.

3.2 Material Datasets

A crucial aspect to manipulate the material properties is to have databases that gather computational models of real-world appearances. Early image-based material datasets include CURet [51], KTH-TIPS [52], or FMD [53]. Surfaces [54] includes over 20,000 real-world images, with surface properties annotated via crowd-sourcing. Also, there

have been efforts on designing common languages for the description of materials such as MDL (Material Definition Language) [55]. Great efforts devoted to measuring BRDFs from real materials have been inverted. Databases with measured materials include MERL [56] for isotropic materials, UTIA [57] for anisotropic ones, the Objects under Natural Illumination Database [58], which includes calibrated HDR information, or the database by Dupuy and Jakob which measures spectral reflectance [59]. In this master's thesis we leverage existing dataset [49, 2] with high-level perpetual rates to develop and train our deep-learning-based framework for single-image material editing.

3.3 Material Editing

Directly editing the parameters of an underlying material model is the simplest form of material editing, for example, manipulating the reflectance of a BRDF. Unfortunately, this approach is unintuitive since it requires expert knowledge, it has many uncorrelated variables and usually yields unpredicted results. Over the years, several perceptually-based frameworks have been proposed to provide more intuitive controls over appearance models. Some works [60, 36] propose psychophysically-based parametric models of surface gloss taken as baseline experiments that explore the relationships between the physical dimensions of glossy reflectance and the perceptual dimensions of glossy appearance. Barla et al. [61] introduce a parametric BRDF model for the rendering of materials that exhibit hazy glossy reflections. However, non-parametric models such as measured BRDFs are harder to edit. Lagunas' thesis [62] proposes intuitive applications to manipulate visual appearance for both material and illumination, while studying the effect of confounding factors on our perception of material appearance. Perceptual editing is a complex task since there is no relationship between our perception and the materials' physical attributes [63, 64, 32]. Some works propose fitting non-parametric BRDFs to parametric models [65], inverse shading trees [66], polynomial bases [66] or using deep-learning techniques [67]. Other works [68, 48, 56] propose to link our perception with non-parametric BRDFs and drive the material editing by human judgments. However, these methods do not allow us to edit the visual appearance without geometrical or illumination information of the scene. We refer the interested readers to the review by Schmidt et al. [69] that provides a comprehensive survey of artistic appearance, lighting and material editing approaches. Our work, proposes a framework that allows editing a single image, in the wild, without any additional information about the scene.

3.3.1 Scene Decomposition Methods

Since scene parameters like geometry and illumination play an important role in our visual perception of the objects [70], some works have been devoted to decomposing the scene into materials, illumination and geometry and manipulating each of them separately [71, 72]. This process is called inverse rendering and usually is a complex problem even for experts in appearance editing. Haber et al. [73] propose a method for recovering the reflectance from a collection of images taken under unknown illumination, allowing us to relight the scene using real-world illuminations. On the other hand, the work from Garces et al. [74] attempts to decompose a single image into

shading and reflectance. Ono et al. [75] present a practical method for reconstructing the bidirectional reflectance distribution function (BRDF) from multiple images of a real object composed of a homogeneous material. Deep learning approaches have been also proposed to segment the scene parameters given a single image [76, 77]. Other works [78, 79, 80] propose deep representations of appearance, i.e. ANNs that learn the relation of color, surface orientation, viewer position, material and illumination to re-render new scenes. Recently, the emergence and popularization of NeRF [81] allow to capture and model a 3D scene from several photographs and the 3D location of their cameras. NeRF has enabled frameworks with unprecedented levels of realism, where following works like NeRD [82] or NeRFactor [83] allow us to decompose the scene into geometry and materials. Other variants like NeRV [84] provides a framework for relighting the scene by changing the 3D position of the light sources. Nevertheless, those methods only provide a new material definition that can later be used in a specific 3D scene but do not allow for modifying the material directly in an existing image. In this work, we develop a generative deep learning that allows us to edit images without the need to solve an inverse rendering problem. Our framework is similar to them since we directly manipulate the pixel values introducing high-level perceptual information about the target look (material).

3.4 A Generative Model for Single-image Appearance Editing

Generative models have demonstrated an outstanding performance to address image editing problems [5, 4, 85, 3]. Recent works even aim to train models for image editing in a total unsupervised manner [86, 87]. The most recent technique from Delanoy et al. [2], introduces a data-driven framework for single-image appearance editing using high-level perceptual attributes.

This framework is composed of two convolutional encoder-decoder generators $\mathcal{G}_i$ with $i \in \{1, 2\}$. To preserve the object's shape the decoders of both generators receive geometrical information from the normal map n of the object, a vector field that expresses how the normal vector varies throughout its surface. $\mathcal{G}_1$ takes as input a low-resolution image of a single object $\mathcal{I}$ together with a high-level perceptual attribute (e.g glossy, metallic, soft, ...) $a \in [-1, 1]$ that describes its appearance. Then the generator compresses the input image in a latent code z_1. Then $\mathcal{G}_1$ reconstructs an edited image $\widehat{\mathcal{I}_{b,1}}$, without high-frequency details by adding the target high-level perceptual attribute $b \in [-1, 1]$ in z_1. Throughout this process, the decoding module is aided by n. The second generator $\mathcal{G}_2$ is a shallow convolutional encoder-decoder that takes as input a high-resolution version of $\mathcal{I}$ and its high-level perceptual attribute a. It repeats the same behavior as $\mathcal{G}_2$ compressing $\mathcal{I}$ into latent code z_2. Then $\mathcal{G}_2$ reconstructs an edited high-resolution image $\widehat{\mathcal{I}_b}$ introducing the target high-level perceptual attribute b in z_2. Throughout the reconstruction phase, $\mathcal{G}_2$ is aided by the normal map n and $\mathcal{G}_1$, as we commented above. Figure 3.1 illustrates the overall workflow of the framework.

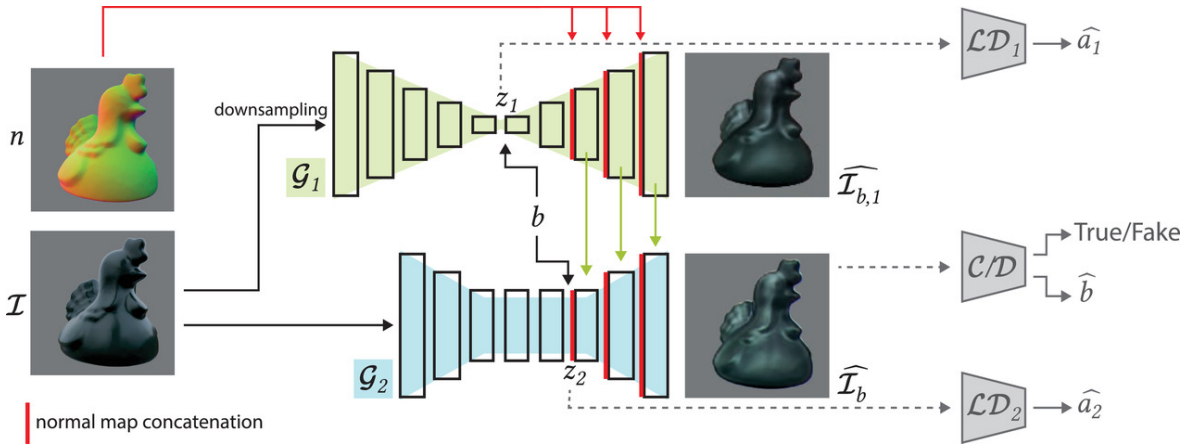


Figure 3.1: Overview of the different components of the framework. Both generators take as input the same image $\mathcal{I}$ but with different resolutions. Red arrows indicate the concatenation between the normal map n and the feature maps of both decoding modules. Green arrows denote the concatenation of the feature maps of $\mathcal{G}_1$ and $\mathcal{G}_2$. The grey-pointed arrows refer to the inputs of the auxiliary discriminator networks only used for training. Image taken from [2].

Since both latent codes, z_i must not contain perceptual information of the high-level perceptual attribute a, during training both generators $\mathcal{G}_i$ play an adversarial game with a latent discriminator $\mathcal{LD}_i$ that tries to predict the values $\hat{a}_i$ of the original high-level perceptual attribute a from the latent codes z_i. Also, the authors introduce an image discriminator $\mathcal{C}/\mathcal{D}$ that plays an adversarial game with $\mathcal{G}_2$ to improve its editing ability. That is, $\mathcal{C}/\mathcal{D}$ attempts to predict the value of the target attribute b from the edited image $\widehat{\mathcal{I}_b}$ while discriminating between real images and fake ones. Both $\mathcal{LD}_i$ and $\mathcal{C}/\mathcal{D}$ are ANNs that are only used to guide the framework during the training phase.

We can separate the editing process into two phases. First, given a certain low-resolution image $\mathcal{I}$ with size of 128×128 px. and its high-level perceptual attribute a, the encoder module of $\mathcal{G}_1$ compresses it into a latent code z_1. Then the desired high-level perceptual attribute b is concatenated to z_1. After that, the normal map n is concatenated to each feature map produced by the convolutional layers of the discriminator, as is illustrated in red in Figure 3.1. These feature maps are stored to help the $\mathcal{G}_2$ in its reconstruction process. The decoder module of $\mathcal{G}_1$ reconstructs the final image $\widehat{\mathcal{I}_{b,1}}$ with the desired look.

In the second phase, $\mathcal{G}_2$ takes the original image $\mathcal{I}$ with size of 256×256 px, repeating the same process as $\mathcal{G}_1$. It encodes the image to a second invariant latent code z_2. Once again, the desired high-level perceptual attribute b is concatenated to z_2 in order to reconstruct the final image $\widehat{\mathcal{I}_b}$. The intermediate features of the pre-trained model $\mathcal{G}_1$, that contain the relevant information on the final target appearance, are concatenated to the decoder's feature maps together with the normal map n, as is illustrated in green in Figure 3.1. The decoder module generates the final image $\widehat{\mathcal{I}_b}$ with higher resolution and preserving the high-frequency details.

Both generators $\mathcal{G}_1$ and $\mathcal{G}_2$ are trained separately, beginning with the training of $\mathcal{G}_1$ from scratch. First, $\mathcal{G}_1$ takes the low-resolution image $\mathcal{I}$ with size 128×128 px; and applies convolution operations repeatedly to obtain the latent code z_i. Since the goal of $\mathcal{LD}_1$ is to predict the ground truth attribute value a from the latent code z_1, it tries to minimize the following distance:

$$\mathcal{L}_{lat}(\mathcal{LD}_1) = ||a - \mathcal{LD}_1(z_1)||_1. \tag{3.1}$$

Where $|| \cdot ||_1$ denotes the L_1-norm. The goal of $\mathcal{G}_1$ is to prevent $\mathcal{LD}_1$ from being able to predict a from z_1 then attempts to minimize the distance:

$$\mathcal{L}_{lat}(\mathcal{G}_1) = -||a - \mathcal{LD}_1(z_1)||_1. \tag{3.2}$$

To improve the reconstruction ability of $\mathcal{G}_1$, the input image is taken as input but now concatenating to z_1 the original high-level perceptual attribute a, then the following error is computed:

$$\mathcal{L}_{rec}(\mathcal{G}_1) = ||\mathcal{I} - \mathcal{G}_1(\mathcal{I}, n, a)||_1. \tag{3.3}$$

$\mathcal{G}_1$ and $\mathcal{LD}_1$ play and adversarial game where $\mathcal{G}_1$ tries to minimize the following adversarial function:

$$\mathcal{L}_{adv}(\mathcal{G}_1) = \mathcal{L}_{rec}(\mathcal{G}_1) + \mathcal{L}_{lat}(\mathcal{G}_1). \tag{3.4}$$

On the other hand, $\mathcal{LD}_1$ tries to minimize the adversarial function:

$$\mathcal{L}_{adv}(\mathcal{LD}_1) = \mathcal{L}_{lat}(\mathcal{LD}_1). \tag{3.5}$$

Once $\mathcal{G}_1$ has been trained, then $\mathcal{G}_2$ learns to edit high-resolution images $\mathcal{I}$'s with size 256×256 px. while keeping the high-frequency details. The generator $\mathcal{G}_2$ repeats the same adversarial training with $\mathcal{LD}_2$ as $\mathcal{G}_1$. Thus, Equations 3.1, 3.2 and 3.3 are optimized by $\mathcal{G}_2$ and $\mathcal{LD}_2$ through the training process as well. An additional discriminator $\mathcal{C}/\mathcal{D}$ is concurrently trained with $\mathcal{G}_2$ in an adversarial game, to improve the generated image quality. The discriminator $\mathcal{D}$ attempts to differentiate between the images belonging to the dataset and those generated by the $\mathcal{G}_2$. The other discriminator $\mathcal{C}$ is trained to predict the attribute value of an image, using the following loss:

$$\mathcal{L}_{lat}(\mathcal{C}) = ||a - \mathcal{C}(\mathcal{I})||_1. \tag{3.6}$$

Meanwhile, the network $\mathcal{G}$ is trained so that the attribute value of the edited image is correctly predicted by $\mathcal{C}$, using:

$$\mathcal{L}_{lat}(\mathcal{C}) = ||b - \mathcal{C}(\mathcal{G}_2(\mathcal{I}, n, b))||_1. \tag{3.7}$$

However, trying to satisfy the attribute predictor can lead $\mathcal{G}_2$ to the generation of unrealistic artifacts in the reconstructed image. Thus, to additionally push the network to generate images that feature the same distribution as the original input data, the

authors introduce a GAN loss together with the image discriminator $\mathcal{D}$. They use the losses from WGAN-GP [88] on both networks $\mathcal{G}_2$ and $\mathcal{G}_1$. The discriminator is trained to rate high the real images and a low the generated ones aiming to disambiguate them, then it tries to minimize the following adversarial function:

$$\mathcal{L}_{adv}(\mathcal{D}) = ||\mathcal{D}(\mathcal{I})||_2 + ||\mathcal{D}(\mathcal{G}(\mathcal{I}, n, b))||_2 + \mathcal{L}_{GP}. \tag{3.8}$$

Where $||\cdot||_2$ denotes the L_2-norm and $\mathcal{L}_{GP}$ the gradient penalty term, we refer the reader to the original work [88] or Section 6.1 for additional information. The generator is trained such that the discriminator believe that generated images are actually real, then attempt to minimize the following adversarial function:

$$\mathcal{L}_{adv}(\mathcal{G}) = -||\mathcal{D}(\mathcal{G}(\mathcal{I}, n, b))||_2. \tag{3.9}$$

Therefore the final adversarial losses are defined as follows:

$$\begin{aligned}
\mathcal{L}_{adv}(\mathcal{G}_2) &= \mathcal{L}_{rec}(\mathcal{G}_2) + \mathcal{L}_{lat}(\mathcal{G}_2) + \mathcal{L}_{adv}(\mathcal{G}_2), && (3.10)\\
\mathcal{L}_{adv}(\mathcal{LD}_2) &= \mathcal{L}_{lat}(\mathcal{LD}_2), && (3.11)\\
\mathcal{L}_{adv}(\mathcal{C}/\mathcal{D}) &= \mathcal{L}_{lat}(\mathcal{C}) + \mathcal{L}_{adv}(\mathcal{D}). && (3.12)
\end{aligned}$$

This workflow, more specifically the training process, is complex and requires a lot of computational resources, and geometrical information is still necessary to preserve geometrical details. This master's thesis presents a novel deep-learning-based method that only needs the original image to model complex visual cues that conform appearance while keeping the high-frequency information from the original image. As we will explain in the following chapter, our method just needs the image together with the target high-level perceptual attribute, that is given by the user, to generate novel edited images. Thus, reducing the model architecture into a single generator helped by an image discriminator with the same behavior as $\mathcal{C}/\mathcal{D}$ during training. Although our model architecture is simpler, we leverage novel layers that allow us to obtain a superior performance in material editing tasks.

Chapter 4

Our Framework

In this chapter, we describe in detail our proposed method for single-image appearance editing. We rely on a synthetic dataset of images paired with high-level perceptual ratings. We use to train a GAN architecture capable of editing material appearance in an intuitive manner. We benign by explaining in Section 4.1 the mathematical model of Selective Transfer Units (STU) memory cells. Then in Section 4.2 we proceed to explain the designed GAN-like model architecture for single-image appearance editing.

4.1 Selective Transfer Units

Sending information between the encoder and the decoder is a common practice in image-translation problems to help the decoder to reconstruct the target output [1]. Usually, the information is sent via skip connections that concatenate the feature maps generated by the encoder with those of the decoder. However, this affects the editing ability of the model, so should be done selectively. That is removing unnecessary information inside the encoder feature maps before sending them to the decoder. In image editing problems, skip connections hamper the editability of the model since it only will be able to reconstruct the input image [2]. One potential solution in encoder-decoder architectures is using the mathematical model of Gated Recurrent Unit (GRU) [89, 90] to memorize information from the encoding phase. Nevertheless, this model has been designed to work with sequential data types, when working with images we need to process information with CNN-like models. The STU memory cell proposed by Liu et al. [3], illustrated in Figure 4.1, is a variant of the GRU [89, 90] model that allows CNNs to memorize information from the encoding phase while removing unnecessary data in the feature maps sent to the decoder.

The STU memory cell retains the relevant information over a long period. Let's say we want to send the feature map of the l^{th} encoder layer denoted by $\mathbf{f}_{enc}^{l}$ to the l^{th} layer of the decoder. Given a single STU, as is shown in Figure 4.1, the hidden state $\hat{\mathbf{s}}^{\mathbf{l+1}}$ holds the information for the next-inner STU cell of the $l+1$ layer. This input is used to remove information from $\mathbf{f}_{enc}^{l}$ an generate the output feature map $\mathbf{f}_{t}^{l}$ as is shown in Figure 4.1. Also the hidden state $\mathbf{s}^{\mathbf{l+1}}$ of the cell is calculated and sent to the previous-outer layer $l-1$.

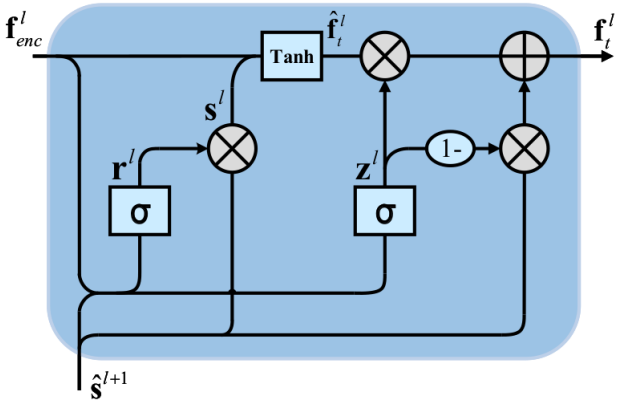


Figure 4.1: Architecture a single STU cell. The inputs are the feature map of the current layer and the hidden state of the next-inner layer. The outputs are the hidden state $\mathbf{s}^l$ and the up-to-date feature map $\mathbf{f}_t^l$. Image taken from [3].

Formally, two tensors $\mathbf{z}$ and $\mathbf{r}$ called update gate and reset gate respectively, decide what information should be passed to the output. The update gate $\mathbf{z}$ helps the model to determine how much of the past information (from the next-inner layer) needs to be passed along to the future. The variable $\mathbf{z}^l$ denotes the update gate for the l^{th} layer and is calculated as:

$$\mathbf{z}^l = \sigma\left(\mathbf{W}_z * \left[\mathbf{f}_{enc}, \hat{\mathbf{s}}^{l+1}\right]\right). \tag{4.1}$$

Where $*$ and $[\cdot,\cdot]$ denote the convolution and concatenation operators respectively. When $\mathbf{f}_{enc}$ is plugged into the network unit, it is concatenated with $\hat{\mathbf{s}}^{l+1}$ and convolved by the tensor $\mathbf{W}_z$. A sigmoid activation function $\sigma(\cdot)$ is applied to normalize the result between 0 and 1 and is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{4.2}$$

On the other hand, the reset gate $\mathbf{r}$ is used from the model to decide how much of the past information to forget. The variable $\mathbf{r}^l$ denotes the reset gate for the l^{th} layer, and is defined as:

$$\mathbf{r}^l = \sigma\left(\mathbf{W}_r * \left[\mathbf{f}_{enc}, \hat{\mathbf{s}}^{l+1}\right]\right). \tag{4.3}$$

This formula is the same as the one for the update gate $\mathbf{z}$. The difference comes in the weights $\mathbf{W}_r$ and the gate's usage. The memory content $\mathbf{s}^l$ uses $\mathbf{r}^l$ to store the relevant information from the past. It is calculated as follows:

$$\mathbf{s}^l = \mathbf{r}^l \circ \hat{\mathbf{s}}^{l+1}. \tag{4.4}$$

Where $\circ$ expresses the Hadamard product. This operation between $\mathbf{r}^l$ and $\hat{\mathbf{s}}^{l+1}$, determines what to remove from the next-inner layer $l+1$. The cell needs to calculate the feature map $\mathbf{f}_t^l$ which holds information from the encoding phase and passes it to the decoder. First, non-linearity is introduced in the form of *tanh* to ensure that the

values in the candidate feature map $\hat{\mathbf{f}}_t^l$ remain in the interval $[-1, 1]$. Then $\mathbf{f}_t^l$ is defined as:

$$\hat{\mathbf{f}}_t^l = \tanh\left(\mathbf{W}_h * [\mathbf{f}_{enc}, \mathbf{s}^l]\right). \tag{4.5}$$

Finally, the update gate $\mathbf{z}$ is needed to determine what to collect from the candidate feature map $\hat{\mathbf{f}}_t^l$ and the memory content $\mathbf{s}^l$. Then, the output feature map $\mathbf{f}_t^l$ is calculated as follows:

$$\mathbf{f}_t^l = (1 - \mathbf{z}^l) \circ \hat{\mathbf{s}}^{l+1} + \mathbf{z}^l \circ \hat{\mathbf{f}}_t^l. \tag{4.6}$$

As in Equations 4.1 and 4.3, the result is convolved by the weights $\mathbf{W}_h$. The Hadamard product allow us to remove the useless information from both $\hat{\mathbf{f}}_t^l$ and $\mathbf{s}^l$. In comparison to GRU [89, 90], where just $\mathbf{f}_t^l$ is adopted as output, we take both the hidden state $\mathbf{s}^l$ and the transformed encoder feature map $\mathbf{f}_t^l$ as output.

4.2 Model Architecture and Formulation

Single-image editing without losing high-frequency details is a non-trivial task. In the previous method [2] described in Section 3.4, the authors use two convolutional encoder-decoder architectures to translate the original image into the desired edited image, but their framework has several limitations. Thus we present a novel generative framework to solve the problems presented in the previous work [2].

As we commented in Section 3.4, the previous method [2] produces blurry results and additionally needs geometrical information from a normal map. Therefore the geometry of the final edited image is strongly related to the input normal map. This causes that a bad estimation of the normal map produces deformations in the geometry of the object in the edited image. This master's thesis studies the possibility of removing the normal map from the input and sending relevant information from the encoder to the decoder with skips connections between features maps following the architecture of U-Net [1]. Unfortunately, directly adding skip connections affects to a large extent the network's editing ability and produces images with no perceptual changes in comparison with the input. Thus, we propose a novel network architecture introducing STU cells at each skip connection between the encoder to the decoder, passing information from inner layers to outer layers.

Our framework is composed of a generator G and a discriminator D with the same number of convolutional layers n. The generator G takes as input a high-resolution image of a single object $\mathbf{x}$ together with a high-level perceptual attribute (e.g glossy, metallic, soft, ...) $\mathbf{att}_s \in [0, 1]$ that describes its appearance. Then the encoder module G_{enc} compresses the input image in a latent code z_{lt}. After that, the decoder module G_{dec} reconstructs an edited image $\mathbf{y}$ by adding the target high-level perceptual attribute $\mathbf{att}_t \in [0, 1]$ in z_{lt}. The features maps sent from the G_{enc} to G_{dec} are processed by the memory module G_{st}.

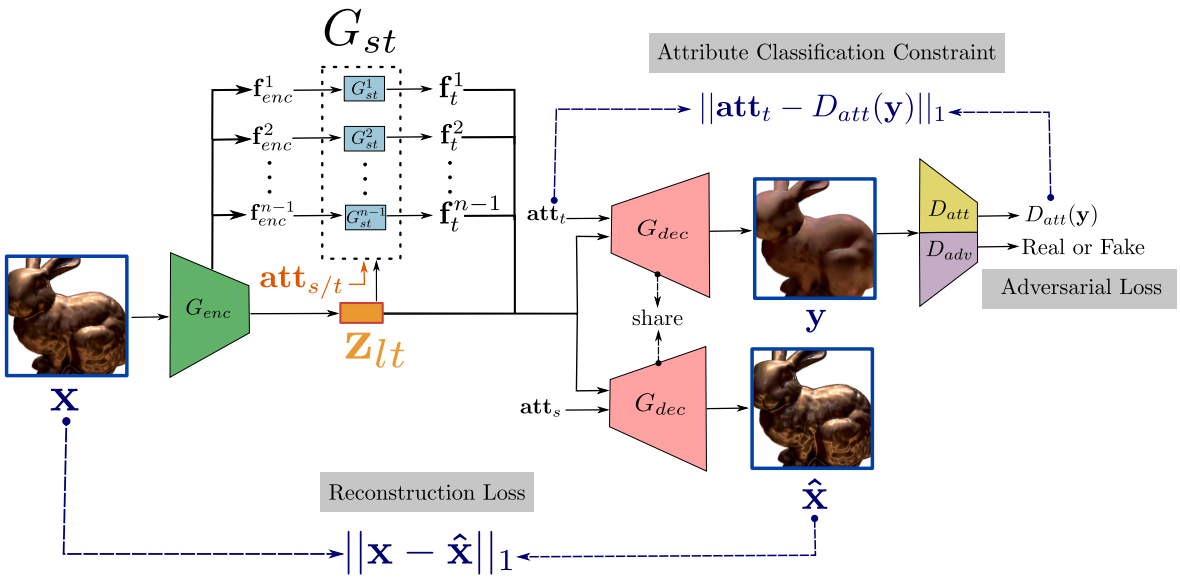


Figure 4.2: Our proposed framework. The green-painted module is the generator's encoder G_{enc} while the red-painted modules represent the reconstructions and edited outputs by the decoder G_{dec}. The black-pointed box is the set of STU cells G_{st}, which edit the feature maps between the encoder G_{enc} and the decoder G_{dec} at each layer. Yellow and purple modules are the branches of the image discriminator D for the adversarial training. The blue-pointed arrows express the values for computing the reconstruction and attribution-manipulation losses explained in Chapter 6.

Because there are no samples of target images, we adopt the adversarial training proposed by He et al. [4]. Thus the discriminator D has two branches D_{adv} and D_{att}. D_{adv} consists of five convolution layers to distinguish whether an image is a fake image or a real one. D_{att} shares the convolution layers with D_{adv} but predicts the high-level attribute value $\mathbf{att}_t$. Figure 4.2 illustrates an overview of our framework.

Formally, the problem may be expressed as an image-to-image translation problem, where we want to transform images on the domain X to another domain Y. Our image generator is in fact performing a mapping $G : X \rightarrow Y$, such that $\mathbf{y} = G(\mathbf{x})$, $\mathbf{x} \in X$ and $\mathbf{y} \in Y$. Given an input image $\mathbf{x}$, the encoder G_{enc} compresses it into the latent code $\mathbf{z}_{lt}$ and stores the set of feature maps $\mathbf{f} = \{\mathbf{f}^1_{enc}, \mathbf{f}^2_{enc}, ..., \mathbf{f}^n_{enc}\}$, where n is the total number of layers that compose of the encoder G_{enc}. As we mentioned above, the variable $\mathbf{f}^l_{enc}$ denotes the feature map of the l^{th} layer of the encoder G_{enc}. The latent code $\mathbf{z}_{lt}$ corresponds to the feature map generated by the last convolutional layer so that $\mathbf{z}_{lt} = \mathbf{f}^n_{enc}$. The set of feature maps $\mathbf{f}$ and the latent code $\mathbf{z}_{lt}$ can be obtained by:

$$(\mathbf{f}, \mathbf{z}_{lt}) = G_{enc}(\mathbf{x}). \tag{4.7}$$

Taken $\mathbf{z}_{lt}$ as the deepest hidden state, this is introduced as input for the STU cell G^{n-1}_{st} of the second deepest layer $n-1$. Each layer hidden state $\mathbf{s}^l$ is re-scaled and concatenated to $\mathbf{att}_t$ before passing it to the previous-outer STU cell to guide the feature updating by the desired perceptual information. Then $\hat{\mathbf{s}}^l$ from the l^{th} layer is defined as:

$$\hat{\mathbf{s}}^l = \mathbf{W}_t *_U \left[\mathbf{s}^l.\mathbf{att}_t\right]. \tag{4.8}$$

Where $*_U$ denotes the upsampling operator defined in Section 2.2.2. We use this instead of a transpose convolution $*_T$ because the latter may produce artifacts on the target image. The result is convoluted by its weight matrix $\mathbf{W}_t$. Then, guided by $\mathbf{att}_t$, STUs are implemented to transform each encoder features $\mathbf{f}^l_{enc}$ into another feature map $\mathbf{f}^l_t$ as:

$$(\mathbf{f}^l_t, \mathbf{s}^l) = G^l_{st}\left(\mathbf{f}^l_{enc}, s^{l+1}, \mathbf{att}_t\right). \tag{4.9}$$

Where G^l_{st} expresses the STU cell of the l^{th} layer, thus do not share parameters with other STUs since the dimensions are different and the features of inner layers are more abstract than those of the outer layers. With $\mathbf{f}_t = \left\{\mathbf{f}^1_t, \mathbf{f}^2_t, ..., \mathbf{f}^{n-1}_t\right\}$ the features maps transformed by the STU in each layer. The target high-level perceptual attribute $\mathbf{att}_t$ is concatenated to $\mathbf{z}_{lt}$ to reconstruct the final edited image $\mathbf{y}$. Each transformed feature map $\mathbf{f}^l_t$ is concatenated with the feature map of the l^{th} layer of the decoder G_{dec} to help to reconstruct the edited image $\mathbf{y}$ expressed as:

$$\mathbf{y} = G_{dec}(\mathbf{z}^{lt}, \mathbf{f}_t, \mathbf{att}_t). \tag{4.10}$$

Considering both G_{enc} and G_{dec}, the output of the generator G can be written as:

$$\mathbf{y} = G(\mathbf{x}, \mathbf{att}_t). \tag{4.11}$$

Chapter 5

Dataset

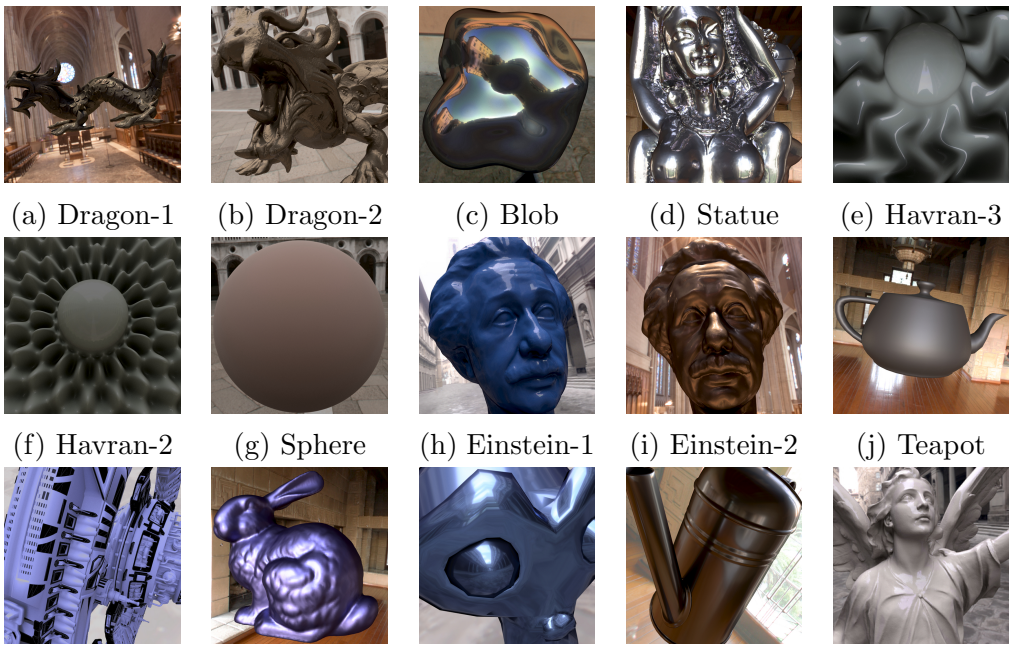
This chapter describes the two datasets used to train and evaluate our proposed framework (Chapter 4). Section 5.1 describes the dataset composed of images and how we collect their high-level perceptual ratings to train our framework. Then, in Section 5.1 we describe the test dataset used to evaluate the reconstruction and editing ability of our framework.

5.1 Training Dataset

The model should generalize well and edit real images under unknown illumination, thus we need a large number of images with different shapes and real appearances. Moreover, each image should be rated by high-level perceptual attributes describing appearance that be used as ground truth in the reconstruction step. Since the ratings are high-level perceptual attributes, those values need to be subjective data collected through subject responses.

Each training image $\mathbf{x}$ is labeled by a set of perceptual high-level ratings of attributes $\mathbf{att}_s$ that describes its appearance. Therefore, the pairs $(\mathbf{x}, \mathbf{att}_s)$ are used to train our framework. We train it using four different datasets to evaluate the performance of each of them (Chapter 7). We take as our basis the extended version of the dataset by Lagunas et al. [49] (denoted as *LD*) and the other three are variants (*MED*, *AVE*, *MI*) of it that we describe below.

The *LD* dataset contains renderings of 13 different geometries (with an additional camera viewpoint for two of them) illuminated by 6 captured real-world illuminations. Renders have been made with the physically-based renderer Mitsuba [91] using 100 different BRDFs (materials) from the Merl dataset [56]. Figure 5.1 illustrates a subset of the dataset with all the geometries. In total, there are 9,000 scenes rendered over four different viewpoints, so the dataset contains a total of 36,000 images.



(a) Dragon-1 (b) Dragon-2 (c) Blob (d) Statue (e) Havran-3

(f) Havran-2 (g) Sphere (h) Einstein-1 (i) Einstein-2 (j) Teapot

(k) Zenith (l) Bunny (m) Suzanne (n) Waterpot (o) Lucy

Figure 5.1: Set of the 13 geometries present on the *LD* dataset, where Dragon and Einstein have been rendered with two different points of view (for a total of 15 scenes).

5.1.1 Subjective attributes

The *LD* dataset has perceptual attributes collected by user studies, those values are associated with subjective-human responses. They are in the form of similarity judgments between pairs of images, which is unsuitable for image editing. Therefore we use a variant of the dataset elaborated by Delanoy et al. [2] following the same methodology as Serrano et al. [48].

The authors carry out a perceptual experiment in which, for each image in the Lagunas et al. dataset [49], participants had to rate several high-level attributes (i.e plastic, rubber, metallic, glossy, bright, rough, the strength of the reflections and the sharpness of reflections) on a Likert-type, 1-to-5 scale. To increase the number of examples, the Lagunas et al. dataset [49] was augmented by creating, for each combination of material $\times$ shape $\times$ illumination, five different images with slight variations in the viewpoint (randomly sampled within a 45 degrees cone around the original viewpoint). Figure 5.2 shows an example with 5 scenes of a bunny with different points of view. The second views for Einstein and Dragon were removed, and a new real-world illumination was included in the illumination set, therefore the final size is around 45,000 images (13 geometries $\times$ 100 materials $\times$ 7 illuminations $\times$ 5 views). The authors relied on Amazon Mechanical Turk to collect the ratings. A total of 2,600 paid subjects participated in the study, each of them seeing 15 different random images. Participants in the study had to go through a training session at the beginning of it, and control stimuli were used to reject invalid subjects.

The final dataset gathers one perceptual rating per viewpoint. To reduce the presence of noise and outliers, the perceptual ratings were pooled over viewpoint and shape using the median, more robust to outliers than the mean. In our experiments, we tested the impact of old perceptual ratings (the original *LD* dataset), median (denoted as *MED*) and average (denoted as *AVE*) and the median value pooled over the 5 viewpoints and the shape (denoted as *MI*), then the attribute is different for each material and illumination.

Figure 5.2: Example of the five viewpoints used in the perceptual study on the bunny shape rendered with the Ennis illumination and nylon material from the augmented Lagunas et al. dataset [49].

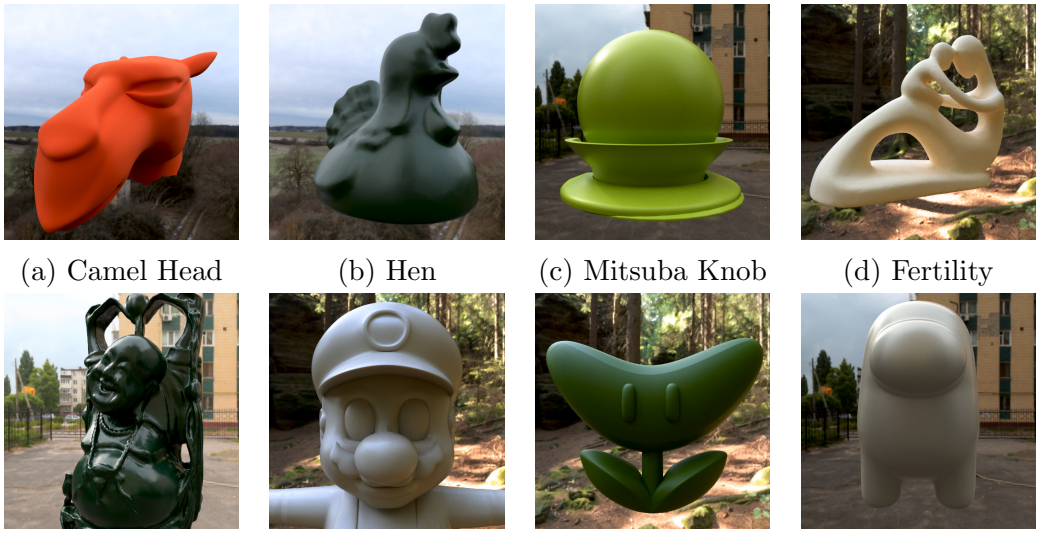
5.2 Test Dataset

To test our models for reconstruction and editing tasks, different classes of images were collected. Images with perceptual ratings collected via user studies are needed to evaluate the reconstruction ability of our models taking the source perceptual attributes as input. So we used the same dataset employed in the previous work to test the networks. This dataset is composed of 53 images shown in Figure B.3. Five unseen geometries illustrated in Figure 5.3, were introduced to with increase the variability of the scenes. A reduced set of graded images was extracted from the training data set to evaluate the model in reconstruction tasks along with some rendered scenes with illuminations not seen during training and the geometries illustrated in the Figure 5.3 that were graded in the same way as the training images.

(a) Suzanne (b) Buddha (c) Camelhead (d) Fertility (e) Cheburaska

Figure 5.3: Representative images of the new geometries used to test the reconstruction performance. The images show 5 unseen geometries included in the reconstruction-test dataset.

On the other hand, the editing ability was tested with non-rated synthetic images and photographs. The first set is conformed of objects with shapes and materials never seen throughout training by our framework. They have been rendered using eight shapes collected from free online sources, four illuminations obtained from



(a) Camel Head (b) Hen (c) Mitsuba Knob (d) Fertility

(e) Buddha (f) Mario Head (g) Flower (h) Among Us

Figure 5.4: Some examples of the synthetic dataset used to test the editing ability of the models. The images show 8 geometries rendered with illumination and materials never seen in the training process.

(a) Metallic Cap (b) Real Buddha (c) Pick Alpaca (d) Seahorse

(e) Tepot (f) Bucket (g) Spoon (h) White Alpaca

Figure 5.5: Scenes present on the set of in-the-wild images used to test the models. The images show scenes with unknown light conditions, geometry, and materials, downloaded from online catalogs.

HDRI-Haven [92], and eight materials coming from Dupuy and Jakob's database [59]. Figure 5.4 shows a representative subset of the synthetic images.

We also need to test our models' performance with real-world images. Thus we collected in-the-wild photographs downloaded from online catalogs of decorative items. Within each image, we masked the object of interest using the online API Kalideo [93]. In Figure 5.5 we can see some examples of the in-the-wild images employed to test the editing ability.

Chapter 6

Training Scheme

In this chapter, we describe the different elements that make up the scheme used to train our proposed framework. First, in Section 6.1 we describe the loss functions optimized throughout the training. Then, in Section 6.2 we give a description of the training of the framework: parameter values, for the loss functions; optimization algorithms and their parameters; hardware specifications; layers in the model's architecture for both generator and discriminator; and the input preprocessing. Finally, we explain our data augmentation strategy to increase variability in the training dataset in Section 6.3.

6.1 Loss Functions

The following sections explain the reconstruction, adversarial, and attribute manipulation losses which are used together to train our model to edit the appearance of the input image while preserving its geometrical details.

Reconstruction loss

We want the generator G to keep the object's shape of the original input image in the edited one. Thus, we introduce a loss to provide feedback to G about its performance keeping the geometry from the input image. As we mentioned in Section 5.1, each input image from the training dataset has a set of high-level perceptual ratings $\mathbf{att}_s$ that describe its appearance. We introduce this variable as input so that the generator learns how to reconstruct the input image generating another image $\hat{\mathbf{x}}$ similar to the input one, which is defined as:

$$\hat{\mathbf{x}} = G(\mathbf{x}, \mathbf{att}_s). \tag{6.1}$$

Therefore our reconstruction loss to be minimized by G is the following L_1-norm:

$$\mathcal{L}_{rec} = ||\mathbf{x} - \hat{\mathbf{x}}||_1. \tag{6.2}$$

Attribute manipulation loss

When the target attributes $\mathbf{att}_t$ are different from $\mathbf{att}_s$, the ground truth of the editing result will be unavailable since we do not have paired original and edited images. Therefore, we use an adversarial loss [94] aiming to edit results indistinguishable from real images. GANs are complex to train, partially due to the instability of the loss function proposed in the original formulation [95]. We rely on the WGAN-GP [88] to alleviate such problems. This loss function employs the Wasserstein distance between distributions and a gradient penalty term $\mathcal{L}_{GP}$. The discriminator is trained to give a high score to real images and a low score to generated ones, aiming to disambiguate them. Its adversarial loss is defined as:

$$\mathcal{L}_{D_{adv}} = \mathbb{E}_{\mathbf{x}} \left[D_{adv}(\mathbf{x}) \right] - \mathbb{E}_{\mathbf{y}} \left[D_{adv}(\mathbf{y}) \right] + \mathcal{L}_{GP}. \tag{6.3}$$

Formally, D should be a 1-Lipschitz continuous function [96]. To keep this constraint, WGANs [95] clips the weights of the discriminator. However, this practice easily breaks the stability and favors the vanishing gradient problem. Meaning that the gradients of the loss function approach zero, making the network hard to train. WGAN-GP replaces weight clipping with a constraint on the gradient norm of the discriminator to enforce Lipschitz continuity. This allows for more stable training of the network than WGAN by adding a gradient penalty term defined as:

$$\mathcal{L}_{GP} = \lambda_1 \mathbb{E}_{\hat{\mathbf{x}}} \left[(||\nabla_{\hat{\mathbf{x}}} D_{adv}(\hat{\mathbf{x}})||_2 - 1)^2 \right]. \tag{6.4}$$

Where λ_1 is the gradient penalty weight (usually 10). Since the expectation $\mathbb{E}$ may be approximated as the L_2-norm, the Equation 6.3 may be written as follows:

$$\mathcal{L}_{D_{adv}} = ||D_{adv}(\mathbf{x})||_2 - ||D_{adv}(\mathbf{y})||_2 + \mathcal{L}_{GP}. \tag{6.5}$$

On the other hand, the generator is trained such that the the discriminator believe that generated images are actually real, giving them a high score. Therefore the generator adversarial loss is given by:

$$\mathcal{L}_{G_{adv}} = \mathbb{E}_{\mathbf{x},\mathbf{att}_t} \left[D_{adv}(G(x, \mathbf{att}_t)) \right], \tag{6.6}$$

As with Equation 6.3, this equation can be rewritten as:

$$\mathcal{L}_{G_{adv}} = ||D_{adv}(G(x, \mathbf{att}_t))||_2. \tag{6.7}$$

Attribute manipulation loss

Even though the ground truth is missing, we need that the editing result has the desired look according to the target perceptual attributes $\mathbf{att}_t$. We introduce an attribute classifier D_{att} which shares the convolution layers with D_{adv} and learns to predict the attribute values $\mathbf{att}_s$ from the images that belong to the training dataset. We compare

the distance from the predicted attributes by D_{att} to $\mathbf{att}_s$ computing the following attribute manipulation loss:

$$\mathcal{L}_{D_{att}} = ||\mathbf{att}_s - D_{att}(\mathbf{x})||_1. \tag{6.8}$$

On the other hand, G should generate images with similar looks as real ones to trick D, so its editings must be consistent with the perceptual attributes $\mathbf{att}_t$. Thus our generator tries to minimize the following distance:

$$\mathcal{L}_{G_{att}} = ||\mathbf{att}_t - D_{att}(\mathbf{y})||_1. \tag{6.9}$$

Final Loss

Taking the above losses into account, the objectives to train the discriminator D and generator G can be formulated as:

$$\min_D \mathcal{L} = -\mathcal{L}_{D_{adv}} + \lambda_2 \mathcal{L}_{D_{att}}, \tag{6.10}$$

$$\min_G \mathcal{L} = -\mathcal{L}_{G_{adv}} + \lambda_3 \mathcal{L}_{G_{att}} + \lambda_4 \mathcal{L}_{rec}, \tag{6.11}$$

where λ_2, λ_3, and λ_4 are the model tradeoff parameters, to work on the same magnitude order as the adversarial losses. Both G and D will try to minimize their objective function, however, once G has learned to trick D, the loss function of the latter will tend to increase until both models reach a stable equilibrium.

6.2 Training Details

This section gives details about the optimization algorithm used to optimize the weights for both G and D; the loss function parameters; and hardware and software specifications. The network architectures of both G and D are detailed in Table 6.1, where we can see the architecture of their convolutional layers.

The learning rate η is a hyper-parameter that we have to tune during training to achieve a better training convergence. The Adaptative Moment Estimation (ADAM) [97] is the optimization method that computes adaptive learning rates for each parameter. It stores an exponentially decaying average of past squared gradients v_t and an exponentially decaying average of past gradients m_t. We compute can compute v_t and m_t for an optimization step t as follows:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla_\theta \mathcal{L}(\theta_{t-1})^2, \tag{6.12}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta \mathcal{L}(\theta_{t-1}). \tag{6.13}$$

Where $\mathcal{L}$ and θ denote the loss function to optimize and the trainable parameters of the model respectively. m_t and v_t are estimates of the first moment (the mean) and

the second moment (the uncentered variance) of the gradients respectively. Both m_t and v_t are initialized as vectors of zeros. The authors observed that they are biased towards zero, especially during the initial time steps and when the decay rates are small (i.e. β_1 and β_2 are close to 1). To bridge this gap, they counteract these biases by computing bias-corrected first and second moment estimates:

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{6.14}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}. \tag{6.15}$$

Then, they use these to update the parameters just as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t. \tag{6.16}$$

Where ϵ is a smoothing term that avoids division by zero (usually on the order of 10^{-8}). To optimize all losses described in Section 6.1, we use $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The learning rate η is 2×10^{-4} for both G and D; and does not decay at training. The tradeoff parameters are $\lambda_2 = 10$, $\lambda_3 = 100$ and $\lambda_4 = 1000$ for the Equations 6.11 and $\lambda_1 = 10$ for Equations 6.4. At each training step, the discriminator iterates seven times while the generator updates its parameters once. Adding skip connections helps the generative models maintain the high-frequency details of the input images. In total, there are four skip connections with STU cells between the encoder and decoder to avoid losing the shape of objects without providing geometrical information.

All the experiments are computed using the Pytorch [6] library with cuDNN 7.1, running on an Nvidia GeForce RTX 3090 GPU. Within each input image, the background is removed by applying a mask of the object's silhouette as is illustrated in Figure 6.1. The target attribute $\mathbf{att}_t$ is sampled randomly by a distribution $\mathbf{U}([0, 1])$ through training. In total, training the whole framework takes two days.

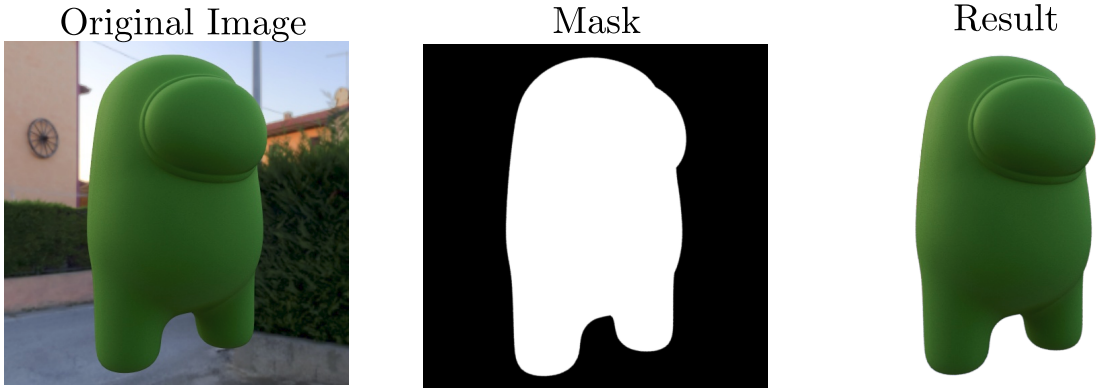


Figure 6.1: Left: A rendered image of the Among Us geometry with a background. Center: Binary mask applied to the input image, black pixels are removed from the input while white pixels remain in the final image. Right: Input image without background as a result of removing pixels from the black area of the mask.

l	G^l_{enc}	G^l_{dec}	D^l_{adv}	D^l_{att}
1	Conv(3,32,4,2), BN, Leaky ReLU	Conv(96, 8,2), BN, ReLU, Conv(8, 3,3,1) , Tanh	Conv(3,32,4,2), IN, Leaky ReLU	
2	Conv(32,64,4,2), BN, Leaky ReLU	Conv(192, 64,3,1), BN, ReLU	Conv(32,64,4,2), IN, Leaky ReLU	
3	Conv(64,128,4,2), BN, Leaky ReLU	Conv(384, 128,3,1), BN, ReLU	Conv(64,128,4,2), IN, Leaky ReLU	
4	Conv(128,256,4,2), BN, Leaky ReLU	Conv(768, 256,3,1), BN, ReLU	Conv(128,256,4,2), IN, Leaky ReLU	
5	Conv(256,512,4,2), BN, Leaky ReLU	Conv(513,512,3,1), BN, ReLU	Conv(256,512,4,2), IN, Leaky ReLU	
			FC(1024), Leaky ReLU	FC(256), Leaky ReLU
			FC(1)	Sigmoid

Table 6.1: Conv(i, o, k, s) denotes the convolutional layer with i input features maps, o output features maps, a kernel of size k and a stride of size s. BN and IN represent Batch Normalization [98] and Instance Normalization [99], respectively. FC(c) refers to a Fully Connected layer with an output of size c.

6.3 Data Augmentation

Deep learning models require large amounts of data to properly work. A common practice is to apply a set of transformations to the input images to provide more synthetic examples to the model. These transformations may be geometrical operations, rotations; image editings, crop or resize; and even channel modifications, increasing the brightness or channel shifts.

To have more diversity in the images and help our model to generalize better, we add a data augmentation pipeline. First, we resize the images to 512 × 512 px; and then they are flipped and rotated 90 degrees randomly, and cropped to 480 × 480 px. To reduce the bias on the colors BRDF we represent images on the HSV color space and make a shift in the Hue and Saturation channels. Finally, the input is resized to a size of 256 × 256 px. to remove the background as is shown in Figure 6.2.

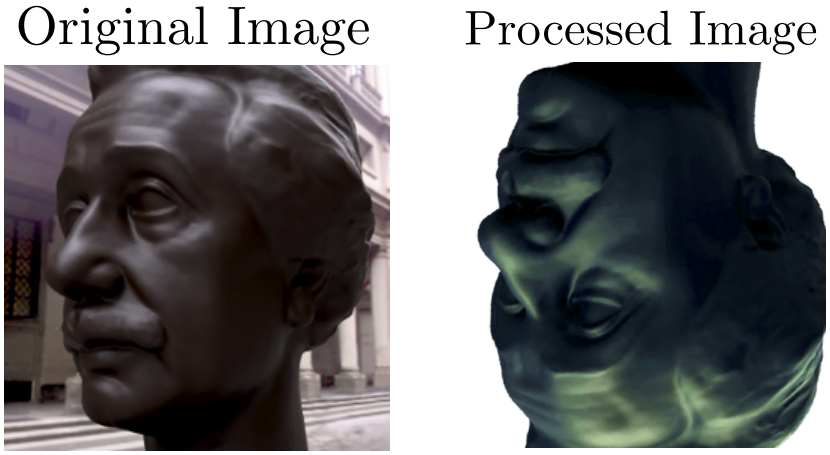


Figure 6.2: Left: A input image of the Einstein geometry with background. Right: Image taken as input after being processed through our data augmentation pipeline. The image has been rotated, cropped, masked and its Hue and Saturation channels in the HSV color space have been shifted.

Chapter 7

Evaluation and Results

This chapter shows our thorough evaluation and our editing results from our proposed framework architecture trained with different statistics of the perceptual attribute values. In Section 7.1 we give a detailed ablation study that validates the effects of training our proposed framework with the different datasets (Section 5.1) for reconstruction and editing tasks. We also explain the bias present in our data and how we deal with it to improve the performance of the framework. Then, in Section 7.2, we explore the consistency of our proposed method in editing tasks, using the test dataset (Section 5.2). Finally, in Section 7.3 we compare our results with the previous method(Section 3.4).

7.1 Ablation Studies

We trained different frameworks on five variants of the dataset described in Section 5.1. Each of them collects one statistic of the high-level perceptual attribute labels. The *LD* dataset is composed of 36,000 renderings with 4 points of view and one attribute per shape, illumination and material combination. We also train our model with the *MED*, *AVE* and *MI* datasets that are variants of the dataset collected by Delanoy et al. [2] that contains 45,000. The *MED* and *AVE* datasets are labeled by the median and average of the perceptual attribute over the 5 points of view respectively. On the other hand in the *MI* dataset, the attribute values are the median value over the 5 viewpoints and the shape, then the attribute is different for each material and illumination.

We evaluate the reconstruction ability of our model trained with the different datasets computing two of the most commonly used measures to compare a pair of images, in our case the input image $\mathbf{x}$ and the reconstructed one $\hat{\mathbf{x}}$. Then, we analyze the editing ability varying the target high-level attribute $\mathbf{att}_t$ for both synthetic and real images. Since we detected bias during our experiments that significantly affects the results, we play with the tradeoff parameters of Equation 6.11 and train a framework with the dataset that has given the best results to counteract this bias.

7.1.1 Reconstruction Metrics

To assess the reconstruction quality of the different models, two common metrics to compare the similarity between images were chosen: Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure [100] (SSIM). Both of them measure the similarity of two images, so we compute them between the input image $\mathbf{x}$ and the generator's reconstruction $\hat{\mathbf{x}}$. The selected metrics are described as follows:

– **PSNR**: It is the ratio between the maximum possible power of an image ($\mathbf{x}$ in our case) and the power of corrupting noise that affects the quality of its representation (noise introduced by G in $\hat{\mathbf{x}}$ into the reconstruction process). To estimate the PSNR of an image, it is necessary to compare that image with an ideal clean image with the maximum possible power. Then the metric is defined as:

$$PSNR(\mathbf{x}, \hat{\mathbf{x}}) = 20 \log_{10}(\max(\mathbf{x})) - 10 \log_{10}(MSE(\mathbf{x}, \hat{\mathbf{x}})). \tag{7.1}$$

Where $\max(\mathbf{x})$ is the maximum value over the pixels of $\mathbf{x}$. In addition we need to compute the Mean Square Error (MSE) between the input $\mathbf{x}$ and the reconstructed image $\hat{\mathbf{x}}$. This error is the difference in the values of pixels of reconstructed image $\hat{\mathbf{x}}$ and the reference image $\mathbf{x}$, both with size of $n \times m$ px; and it is defined as:

$$MSE(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{nm} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (\mathbf{x}[i,j] - \hat{\mathbf{x}}[i,j])^2. \tag{7.2}$$

– **SSIM**: Our visual perception system is highly capable of identifying structural information from a scene, and hence identifying the differences between the information extracted from a reference ($\mathbf{x}$ in our case) and a sample scene ($\hat{\mathbf{x}}$ in our case). This metric proposed by Wang et al. [100], replicates this behavior, therefore allowing us to compute the difference between a noisy and a reference image as:

$$SSIM(\mathbf{x}, \hat{\mathbf{x}}) = [l(\mathbf{x}, \hat{\mathbf{x}})]^{\alpha} \cdot [c(\mathbf{x}, \hat{\mathbf{x}})]^{\beta}. \tag{7.3}$$

The SSIM rely on the illumination I and contrast V of the reference $\mathbf{x}$ and the noisy image $\hat{\mathbf{x}}$. The illumination of an image $\mathbf{z}$ with size $n \times m$ px. may be approximated as the average values of their pixels, so it is defined as:

$$I \approx \mu_{\mathbf{z}} = \frac{1}{nm} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} \mathbf{z}[i,j]. \tag{7.4}$$

The first term l of the Equation 7.3 is the luminance comparison function and is defined as:

$$l(\mathbf{x}, \hat{\mathbf{x}}) = \frac{(2\mu_{\mathbf{x}}\mu_{\hat{\mathbf{x}}} + C_1)}{(\mu_{\mathbf{x}}^2 + \mu_{\hat{\mathbf{x}}}^2 + C_1)}. \tag{7.5}$$

Where $\mu_{\mathbf{x}}$ and $\mu_{\hat{\mathbf{x}}}$ are the illumination of the reference $\mathbf{x}$ and the noisy image $\hat{\mathbf{x}}$ respectively. On the other side, the contrast V of a certain image $\mathbf{z}$ with size $n \times m$ px. is approximated as the standard deviation of all its pixels, so it is defined as:

$$V \approx \sigma_{\mathbf{z}} = \left(\frac{1}{nm} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (\mathbf{z}[i,j] - \mu_{\mathbf{z}})^2 \right)^{\frac{1}{2}}. \tag{7.6}$$

Therefore the second term c on the Equation 7.3 is the contrast comparison function, and is defined as:

$$c(\mathbf{x}, \hat{\mathbf{x}}) = \frac{(2\sigma_{\mathbf{x}}\sigma_{\hat{\mathbf{x}}} + C_2)}{(\sigma_{\mathbf{x}}^2 + \sigma_{\hat{\mathbf{x}}}^2 + C_2)}. \tag{7.7}$$

Where $\sigma_{\mathbf{x}}$ and $\sigma_{\hat{\mathbf{x}}}$ are the contrast of the reference $\mathbf{x}$ and the noisy image $\hat{\mathbf{x}}$ respectively. For our experiments we set $\alpha = 1$, $\beta = 1$, $C_1 = 0.01^2$ and $C_2 = 0.03^2$.

7.1.2 Image Reconstruction

We compare the reconstruction ability of different trained models using the reconstruction test data introduced in Section 5.2. Table 7.1 collects the average reconstruction metrics from generated images by the trained models on different datasets. Figure 7.1 shows some reconstructed images from the test dataset. There, we can see that the framework trained on the *MI* dataset is the framework that performs best in reconstruction tasks.

Training Dataset	**PSNR ↑**	**SSIM ↑**
LD	26.71	0.961
MED	27.15	0.955
AVE	27.61	0.964
MI	**28.32**	**0.970**

Table 7.1: Average PSNR and SSIM from results on Tables A.1 and A.2 in Appendix A. *LD* refers to the original dataset collected by Lagunas et al. [49], which is composed of 36,000 renderings with 4 points of view and one attribute per shape, illumination and material combination. *MED* and *AVG* are the median and average of the perceptual attribute over the 5 points of view of the collected data by Delanoy et al. [2] respectively. In *MI*, the attribute values are the median value over the 5 viewpoints and the shape, then the attribute is different for each material and illumination.

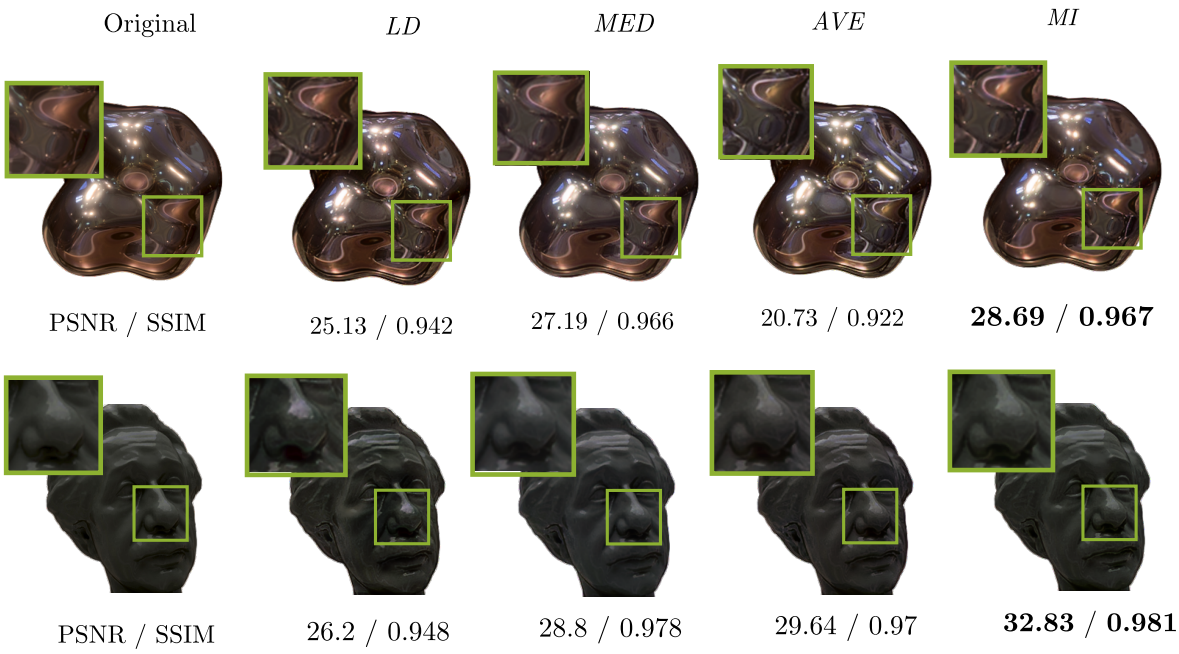


Figure 7.1: Examples of results that illustrate the reconstruction ability of the different editing frameworks trained with the *LD*, *MED*, *AVE* and *MI* datasets. Upper-row: A scene rendered using the Blob geometry. Lower-row: A scene rendered using Einstein geometry. The rows of numbers are the PSNR and SSIM metrics calculated from the reconstruction $\hat{\mathbf{x}}$ generated by each model and the original image $\mathbf{x}$.

7.1.3 Appearance Editing

All tested datasets give a great performance in reconstructing the input image. However, the main challenge consists in editing the high-level perceptual attributes properly. Usually, skip connections reduce the models' editing ability [2]. We test our proposed architecture on synthetic and real datasets shown in Section 5.2, varying the target attributes from 0 to 1.

We train one model per attribute (i.e glossy and metallic) and dataset. In total, eight models were tested, four of them learn to edit metallic perceptual attributes from *LD*, *MED*, *AVE* and *MI* datasets and the others are trained to perform glossy edits. Figure 7.2 shows the editing results varying the metallic and glossy target attribute in in-the-wild images with unknown scene parameters. On the other hand, Figure 7.3 illustrates the editing performance on synthetic images with known scene parameters.

Edited images keep details from the input image and introduce the proper changes in the perceptual cues to achieve the desired look of the target attribute. The trained model for glossy editing has learned where light comes from and increases the brightness correctly for synthetic and real images. On the other side, the frameworks trained for metallic editing faithfully edit images by giving them a distinctive metallic look.

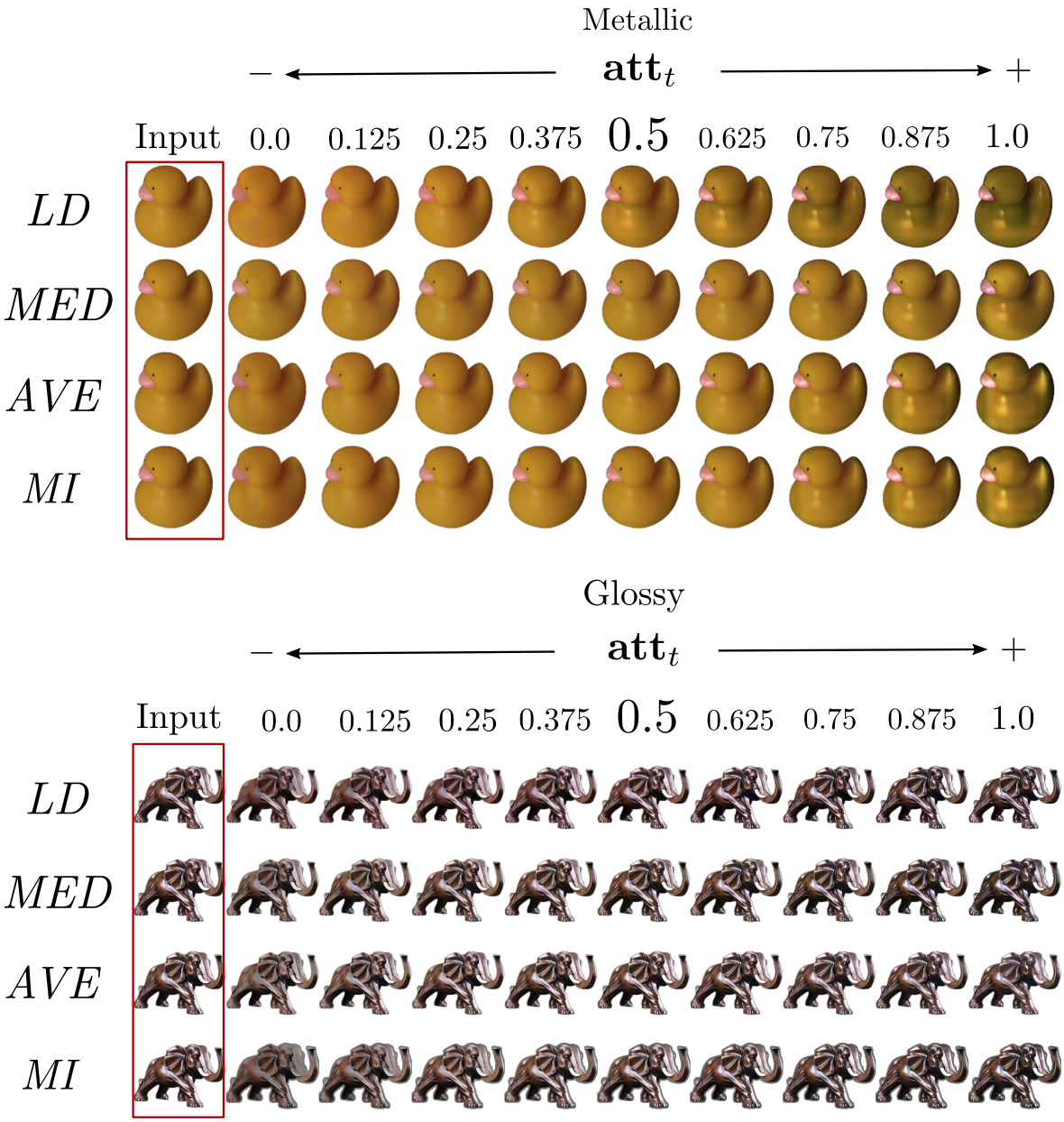


Figure 7.2: Editing results by varying the high-level perceptual attributes metallic and glossy. Each row are the results generated by models trained with different datasets. First column represents the input real image $\mathbf{x}$ while following ones are the edited images sampling the target perceptual attributes $\mathbf{att}_t$ as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$.

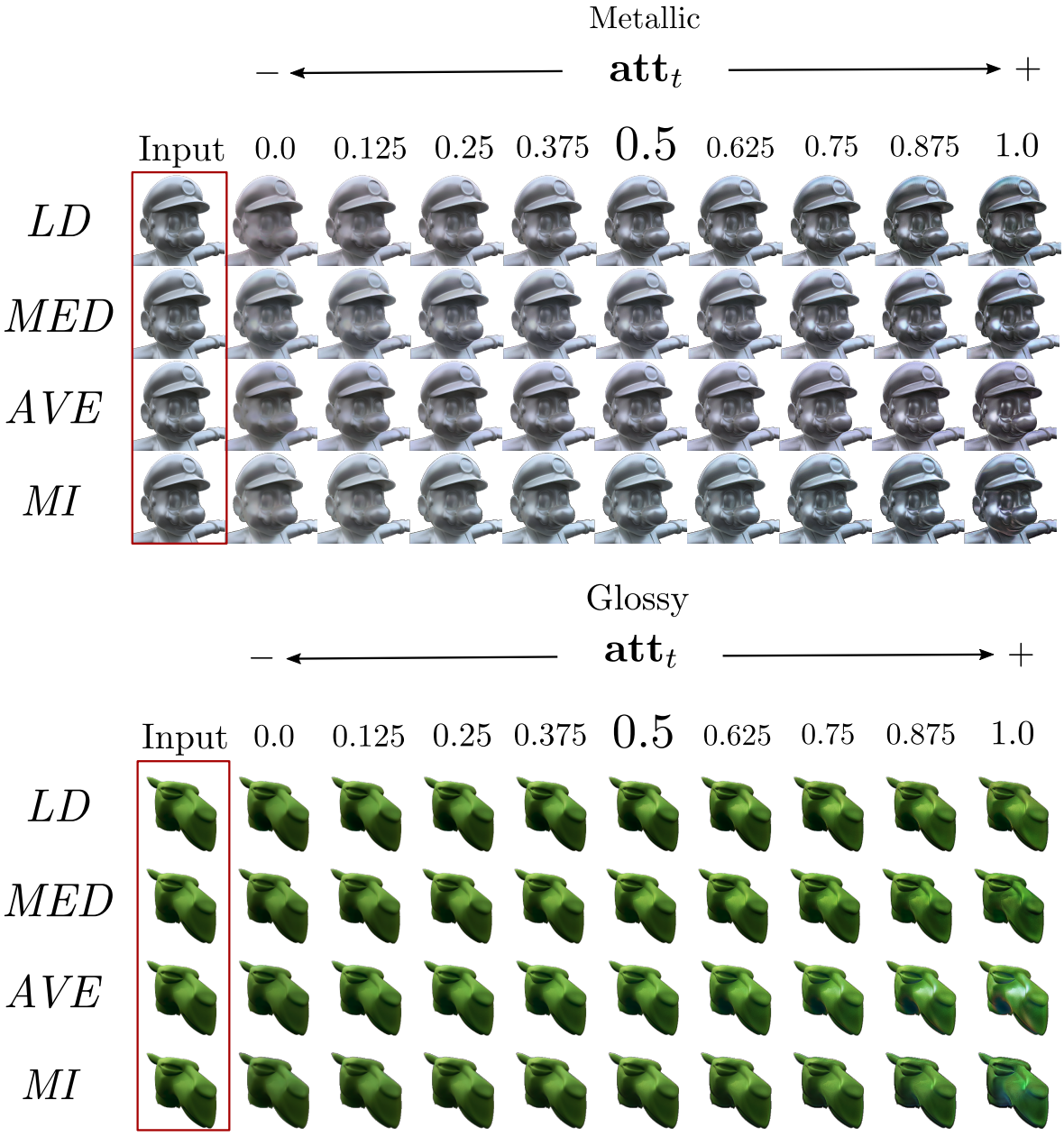


Figure 7.3: Editing results by varying the high-level perceptual attributes metallic and glossy. Each row are the results generated by models trained with different datasets. First column represents the input synthetic image $\mathbf{x}$ while following ones are the edited images sampling the target perceptual attributes $\mathbf{att}_t$ as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$.

7.1.4 Perceptual Bias

Since our data are linked to perceptual judgements, there is a bias between glossy and metallic attributes. Metallic material tends to have high specular properties. This drives the users' decisions to rate metallic images as high glossy objects 7.4, therefore there are no samples jointly rated as non-glossy and metallic or vice-versa.

Figure 7.4: Left: Scenes of two objects of Blob and Bunny geometries. They have been rendered using beige and yellow paint material respectively. Both have been rated by the user with 0 for both glossy and metallic. Right: we can see a dragon and a water pot with aluminum and black obsidian materials respectively. They have been classified as pure glossy and metallic objects by the users, getting a rate of 1 for both attributes.

The generator should increase the brightness regions while keeping the object's color unchanged when it edits the glossy attribute. For metallic editing, both properties should be modified to achieve different looks between glossy and metallic. However, as we can see in Figure 7.5 glossy and metallic edited images looks similar enough.

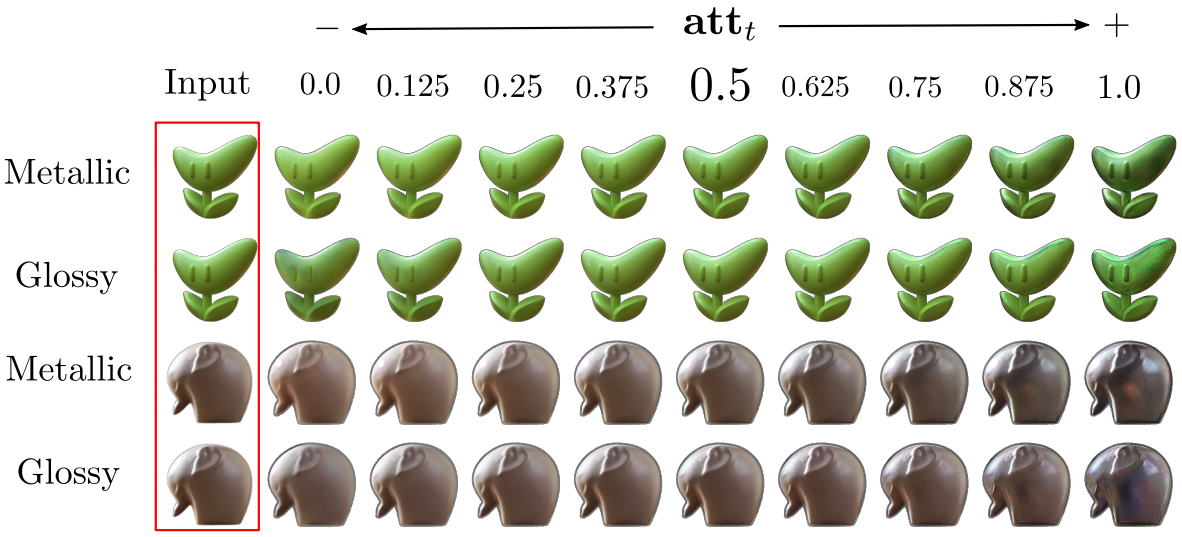


Figure 7.5: Edited images by the model trained on the *MI* dataset varying the target perceptual attributes metallic and glossy. Each pair of rows express the results sampling both target attributes $\mathbf{att}_t$ as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$.

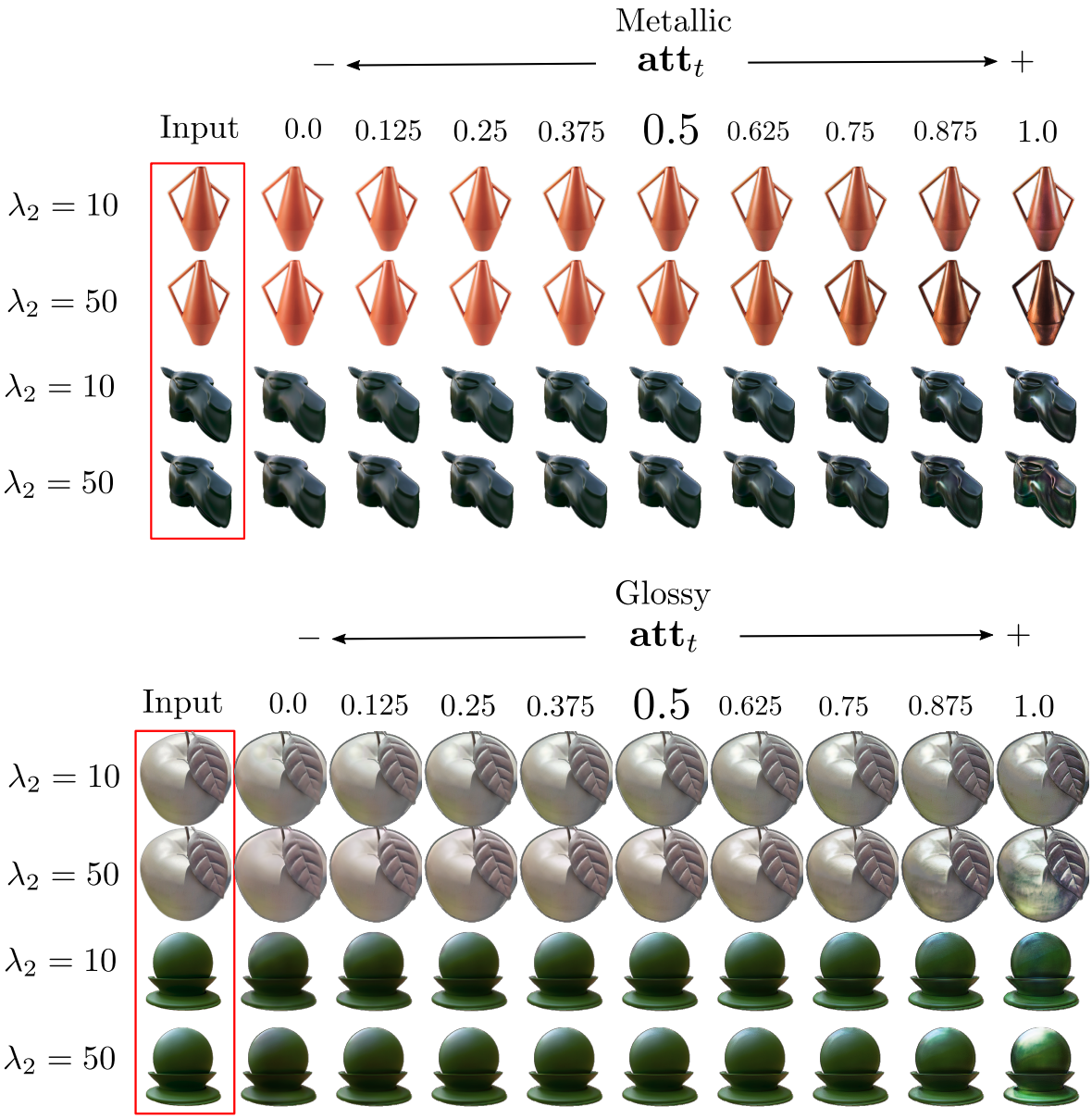


Figure 7.6: Edited images varying the target perceptual attributes metallic and glossy. Each pair of rows are the generated images by models trained on MI dataset, but with different values for λ_2. The target attributes $\mathbf{att}_t$ are sampled as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$ for rendered images and photographs.

Since collecting unbiased data is unfeasible, to bridge this gap, we played with the tradeoff parameters of Equation 6.11 to increase the variability of generated images. Checking the evolution of loss functions during training, we observe that $\mathcal{L}_{D_{adv}}$ and $\mathcal{L}_{D_{att}}$ where on the same scale, but the second one decreases faster than the first one. The parameter λ_2 controls the contribution of the attribute manipulation loss $\mathcal{L}_{D_{att}}$. We raise it to increase editing ability and help our model to distinguish between glossy and metallic. Since the model trained on the *MI* dataset has the best reconstruction results, as is shown in Table 7.1, we re-trained it with $\lambda_2 = 50$ for glossy and metallic attributes. Figure 7.6 shows a comparison between the old parameter and the new one.

7.2 Results

In this section we give the results of our final framework, trained with the *MI* dataset and $\lambda_2 = 50$ and introduced in Section 7.1.4. We train one model per attribute and test both of them using rendered images with different geometries, illuminations and materials; and photographs under unknown scene parameters.

To asses the editing ability of our framework we vary the target high-level perceptual attributes glossy and metallic as is shown in Figure 7.7. The framework changes the object's color together with brightness when the value of the target metallic attribute is high. On the other hand, decreasing the target metallic attribute also keeps the specular reflections instead of removing them.

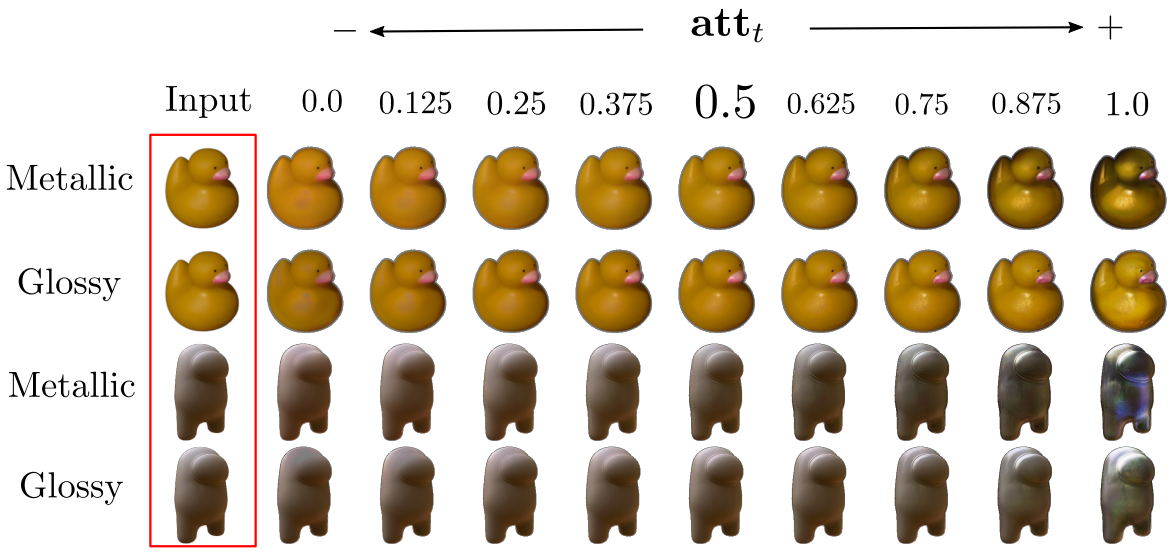


Figure 7.7: Edited images varying the target perceptual attributes metallic and glossy. Each pair of rows shows the results for both attributes $\mathbf{att}_t$ sampled as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$.

We test the consistency of our framework exploring its editing ability through examples from the synthetic dataset. Since we have a wide collection of rendered images with diverse materials, illuminations and geometries, we can fix some scene parameters and test the model by varying the other two parameters. Figures 7.9 and 7.8 show edited images by our framework for glossy and metallic attributes respectively, while varying the scene parameters.

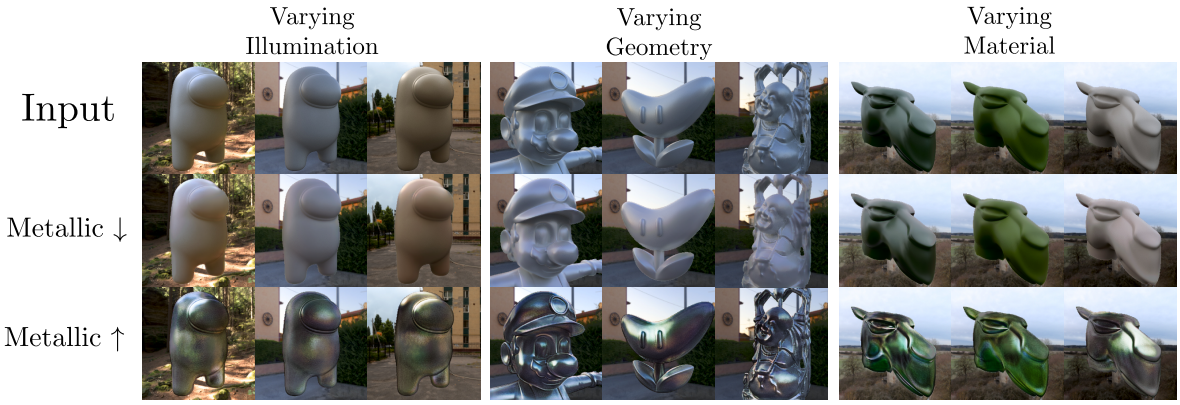


Figure 7.8: Left: Same geometry and material but four different illuminations in the input image. Center: Same illumination and material, but four different geometries. Right: Same geometry and illumination, but two different materials. Our framework is capable of producing compelling and consistent edits in all cases. Arrows pointing up correspond to a target metallic attribute value $\mathbf{att}_t$ of 1, while arrows pointing down correspond to a value of 0.

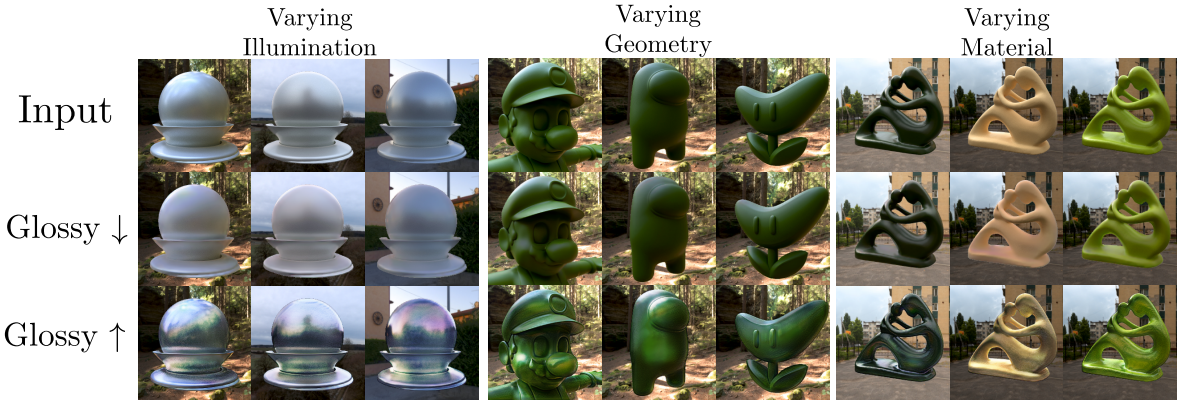


Figure 7.9: Left: Same geometry and material but three different illuminations in the input image. Center: Same illumination and material, but three different geometries. Right: Same geometry and illumination, but three different materials. Our framework is capable of producing compelling and consistent edits in all cases. Arrows pointing up correspond to a target glossy attribute value $\mathbf{att}_t$ of 1, while arrows pointing down correspond to a value of 0.

When working with in-the-wild images, photographs in our case, the scene parameters are completely unknown. Because of this, even though we have a large dataset composed of different complex scenes, we can't modify their material and geometrical parameters as we please. We can only play with the illumination conditions of the scene. Thus, to test the consistency of our framework to edit in-the-wild images we vary the illumination while keeping the geometry and material constant, as we can see in Figure 7.10. We also test the temporal consistency of our framework editing two video sequences frame by frame for both metallic and glossy attributes. More results in Appendix C.

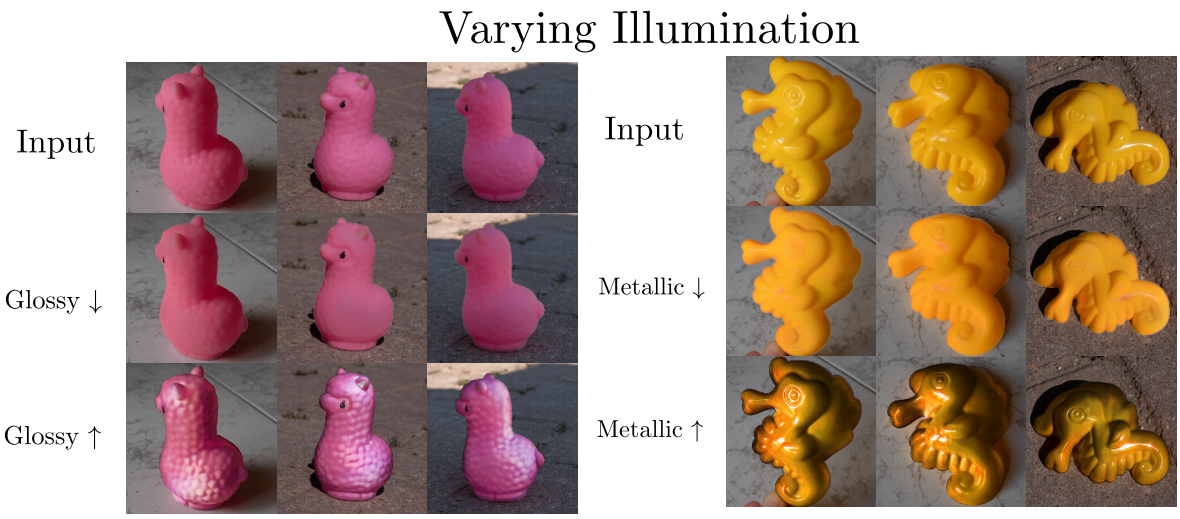


Figure 7.10: Left: Same geometry and material but three different illuminations in the input image while editing the glossy attribute. Right: Same geometry and material but three different illuminations in the input image while editing the metallic attribute. Our framework is capable of producing compelling and consistent edits in all cases. Arrows pointing up correspond to a target attribute value $\mathbf{att}_t$ of 1, while arrows pointing down correspond to a value of 0.

7.3 Comparison with the Previous Method

We compare the model evaluated in Section 7.2, setting $\lambda_2 = 50$, with the previous work explained in Section 3.4. First, we compute the PSNR and SSIM [100] for both methods. Since the framework proposed by Delanoy et al. [2] enters in mode collapse if it is trained with the whole *MI* dataset, for the comparison we use models pre-trained by the authors on a reduced version of the *MI* dataset (*MI-S*) which contains only 9,000 images instead of 45,000. As is shown in Table 7.2, we can see that our model performs better in the reconstruction metrics than the work of Delanoy et al. [2].

Method / Training Dataset	**PSNR ↑**	**SSIM ↑**
Our Framework / *MI*	**27.388**	**0.967**
Delanoy et al. / *MI-S*	18.24	0.826

Table 7.2: Average PSNR and SSIM from results on Tables A.1 and A.2 in Appendix A. The previous work from Delanoy et al. [2] was trained with the *MI-S* dataset, composed of $9,000$ images from the MI dataset, which is a collection of 45,000 renders as is explained in Section 5.1.

As illustrated in Figure 7.11, our method keeps high-frequency details from the input image without the need of a normal map of the object's surface. We can see that specular reflections are present on the reconstructed image while the previous method removes them keeping only low frequencies.

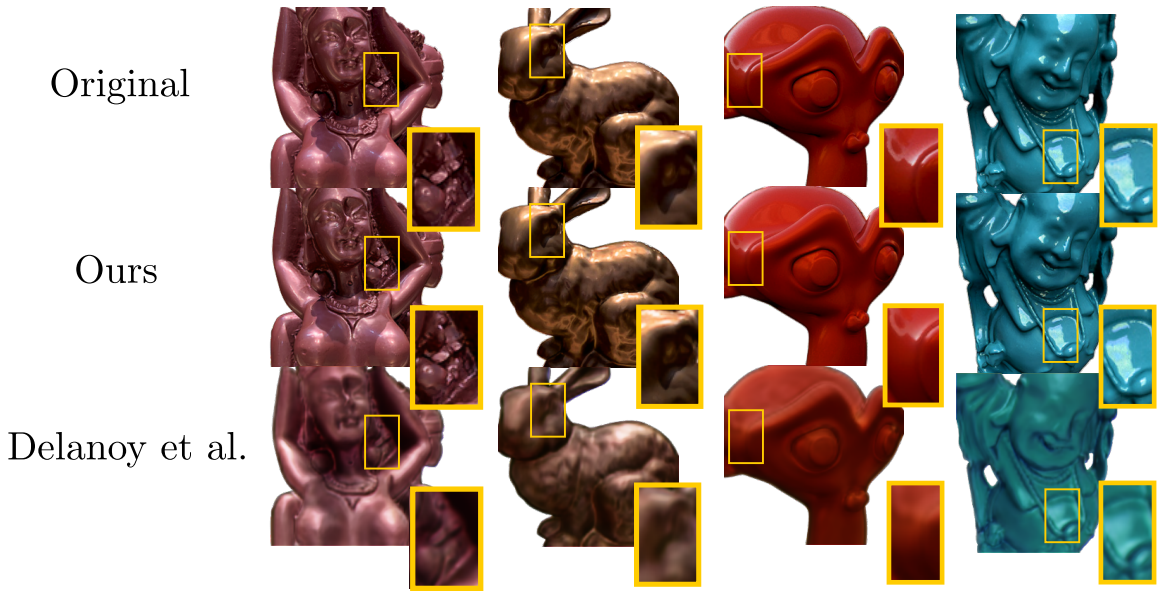


Figure 7.11: A demonstration of the reconstruction quality the two methods for geometries: Statue, Bunny, Suzanne and Buddha. Yellow boxes zoom in regions of the scenes in reconstructed and input images.

In Figure 7.12 we can see a comparison between the edited images by our method and the one by Delanoy et al. [2]. Our approach learns to edit perceptual cues properly while objects' shapes remain unchanged. The geometry of the generated images by the other method, from Delanoy et al. [2], depends strongly on the shape of the normal map, which has been estimated by a normal predictor [101] (another neural network). This causes geometry details that are not present in the normal map to not be present in the output image. Also, inaccurate estimation of the normal map may deform the original shape, especially in in-the-wild images where usually geometries are complex. We can see an example of this in the yellow-plastic duck and the black-cat statue in Figure 7.12. More comparisons can be found in Appendix C.

The number of trainable parameters is an important factor when a deep learning model is being designed because a large number of parameters involves using a lot of memory to train the models and, often, resources are scarce. Our framework is composed of one generator G and one discriminator D, while the previous approach needs two generators $\mathcal{G}_i$ helped by other two latent discriminators $\mathcal{LD}_i$ and one image discriminator $\mathcal{C}/\mathcal{D}$ to improve the editing ability. Since our framework is simpler than the previous model, it has fewer trainable parameters, so we reduce notably the memory usage as is shown in Table 7.3.

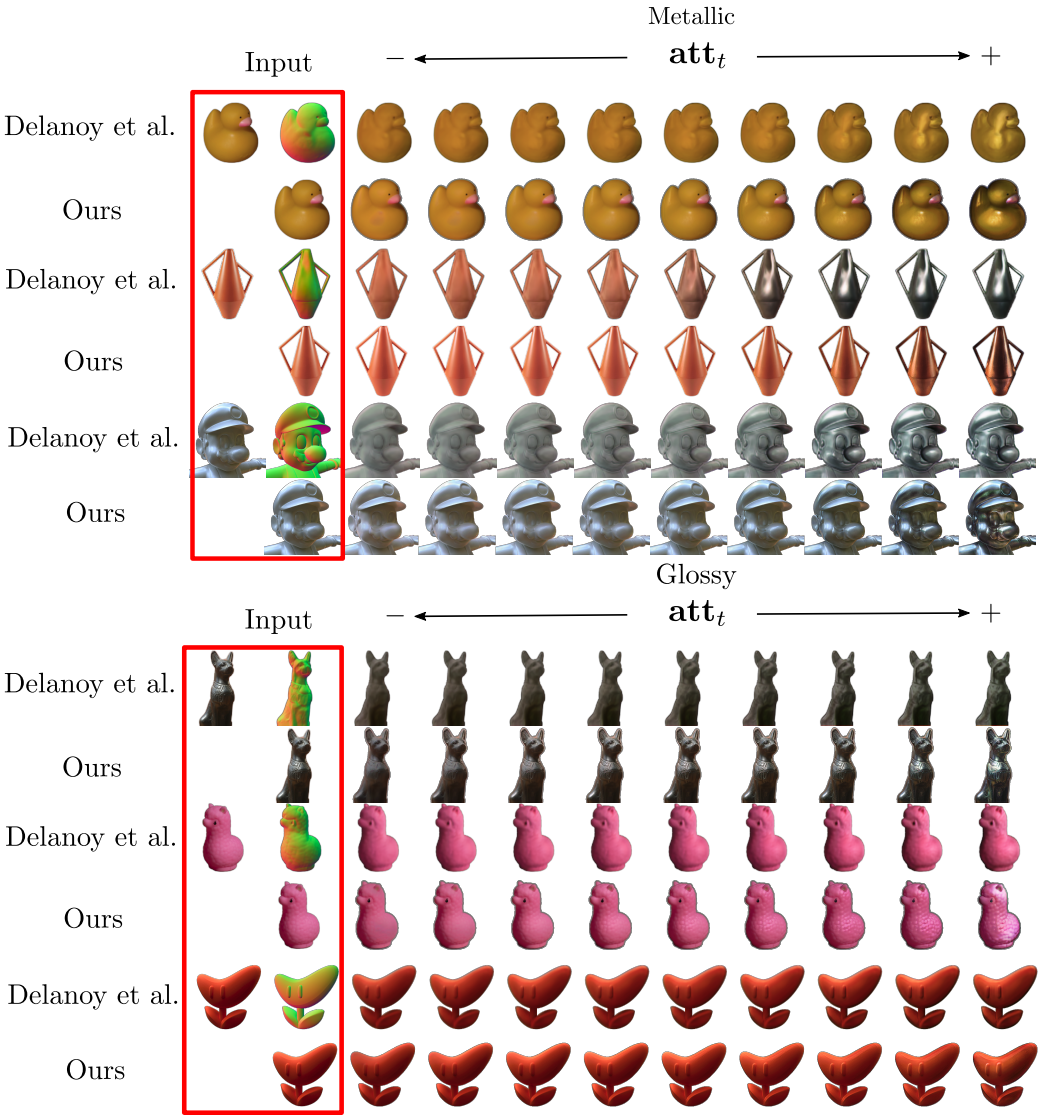


Figure 7.12: Each pair of rows are the results generated by our method and the method proposed by Delanoy et al. [2]. First column represents the input $\mathbf{x}$, while following ones are the edited images sampling the target perceptual attributes $\mathbf{att}_t$ as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$ for our model while for the method of Delanoy et al. [2], attributes are sampled as $\{-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1\}$.

Method	Module	Trainable Parameters	Total Trainable Parameters
Delanoy et al.	$\mathcal{G}_1$	20 356 515	58 920 489
	$\mathcal{LD}_1$	4 326 401	
	$\mathcal{G}_2$	4 610 179	
	$\mathcal{LD}_2$	14 813 697	
	$\mathcal{C}/\mathcal{D}$	14 813 697	
Our Framework	G ($G_{enc} + G_{dec} + G_{st}$)	13 758 280	**33 326 314**
	D (both D_{att} and D_{adv})	19 568 034	

Table 7.3: Number of trainable parameters per module for both methods. They are updated into each backward pass step of the back propagation algorithm.

Chapter 8

Conclusions

This master's thesis has presented a method for perceptual editing directly in the input image, without supplemental information of the scene such as the normal map. The proposed deep-learning-based framework has been validated with many varieties of images. Our experiments explore the performance with geometries, materials and illuminations never seen in the training process. Testing the performance by online images, with unknown illumination conditions, demonstrates how our framework learns to model approximate the complex interactions between light and matter. Moreover, we have seen that our novel architecture allows to obtain superior performance while having less input information (we do not require a normal map) and a simpler, cheaper model.

The work carried out reveals the potential of adding temporal information in image generation tasks based on GAN-like models. Our results in reconstruction tasks show how introducing STU cells in skips connections, help generative models to keep high-frequency information in generated images. The previous method needs to concatenate a normal map to the feature maps of the generator, forcing it to design an elaborate network architecture. Adding an internal-hidden state to each layer on our framework reduces the problem to a simple generator-discriminator architecture.

Our experiments with synthetic images and photographs show how selecting the proper trade-off parameters increase notably the ability of the framework to generate distinguishable glossy and metallic edits. Also, removing normal maps from the input eliminates the geometrical deformations in the edited images caused by an inaccurate normal map estimation. Moreover, the proposed network architecture allows us to remove the latent discriminators and the second image generator that are present in previous work. Thus, the number of trainable parameters have been reduced from 58,920,489 parameters to 33,326,314; saving 43 % of memory.

8.1 Future Work and Limitations

This work is one of the first studies focusing on the exploration of the effect of memory cells in appearance editing tasks with single images and unknown scene parameters. However, there is still room for improvement in in-the-wild and synthetic image editing.

It would be interesting to asses the performance of our framework by including metrics other than those used, or to carry out additional procedures to validate the editing ability of our method.

Including skips connections between the encoder and the decoder helps the model to keep high-frequencies details. Nevertheless, sending information from the encoding phase may confuse the model in the editing of some visual effects present on the object's surface like perfect specular reflections as in show Figure 8.1. Also, the models have been trained with synthetic examples, which do not cover all kinds of light-matter interactions. To solve this problem it would be necessary to analyze in detail the architecture of the STU cells and the impact of the reconstruction loss or increasing the color variability of the train samples modifying the data augmentation pipeline.

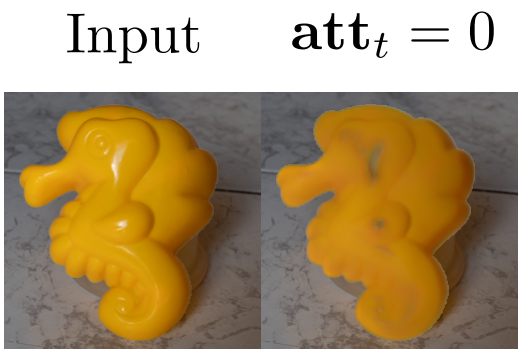


Figure 8.1: Left: Image $\mathbf{x}$ of an object with perfect specular reflections at its surface. Right: edited image $\mathbf{y}$ where the target perceptual attribute glossy is decreased to 0 and the model removes the specular reflections by adding gray blobs.

We train one model per perceptual attribute glossy and metallic. However, it would be interesting training a model with both attributes simultaneously to avoid have one model per attribute. Following this line, one could add other attributes describing appearance (like diffuse, ceramic, etc) and train a model to have an intuitive appearance vector with as many values as appearance attributes. Also, throughout experiments we work with images with size of 128×128 px. due to the time and resource constraints. It would be motivating try to work with higher resolution images (e.g., with size of 512×512 px.).

To assess and compare our frameworks we use the PSNR and SSIM [100] metrics. However, these metrics are not designed to compare color images in particular. Computing other metrics designed for that proposal like iCAM [102] or the one proposed by Grevcova et al. [103], could help to compare the frames more accurately. In addition, it would be interesting to validate the quality of the generated images via human judgments. Replicating the study proposed by Delanoy et al. [2], where the images were rated by groups of non-expert people. Another interesting line of work would be to rely on psychophysical experimentation to evaluate the quality of the results obtained. To test the ability of our model to edit synthetic images, the parameters of the BRDF would be changed manually and then render the scene to compare it with the edited images by our framework.

In view of the promising results obtained, the work developed in this master's thesis aims to be extended, trying to mitigate the problems discussed in this section, and later aiming to publish our contributions in a journal or conference of computer graphics or machine learning (e.g., the Computer Vision and Pattern Recognition conference or the Eurographics conference).

Chapter 9

Bibliography

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[2] J Delanoy, M Lagunas, J Condor, D Gutierrez, and B Masia. A generative framework for image-based editing of material appearance using perceptual attributes. In *Computer Graphics Forum*. Wiley Online Library, 2022.

[3] Ming Liu, Yukang Ding, Min Xia, Xiao Liu, Errui Ding, Wangmeng Zuo, and Shilei Wen. Stgan: A unified selective transfer network for arbitrary image attribute editing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3673–3682, 2019.

[4] Zhenliang He, Wangmeng Zuo, Meina Kan, Shiguang Shan, and Xilin Chen. Arbitrary facial attribute editing: Only change what you want. *arXiv preprint arXiv:1711.10678*, 1(3), 2017.

[5] Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc'Aurelio Ranzato. Fader networks: Manipulating images by sliding attributes. *Advances in neural information processing systems*, 30, 2017.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[8] Cs354. https://www.cs.utexas.edu/~theshark/courses/cs354/.

[9] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986.

[10] Ibón Guillén, Julio Marco, Diego Gutierrez, Wenzel Jakob, and Adrian Jarabo. A general framework for pearlescent materials. *ACM Transactions on Graphics*, 39(6), 2020.

[11] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[12] Peter Shirley. Ray tracing in one weekend, December 2020. https://raytracing.github.io/books/RayTracingInOneWeekend.html.

[13] Peter Shirley. Ray tracing: The next week, December 2020. https://raytracing.github.io/books/RayTracingTheNextWeek.html.

[14] Peter Shirley. Ray tracing: The rest of your life, December 2020. https://raytracing.github.io/books/RayTracingTheRestOfYourLife.html.

[15] Rosana Montes and Carlos Ureña. An overview of brdf models. *University of Grenada, Technical Report LSI-2012-001*, 2012.

[16] Omar Costilla-Reyes, Ruben Vera-Rodriguez, Abdullah S Alharthi, Syed U Yunas, and Krikor B Ozanyan. Deep learning in gait analysis for security and healthcare. In *Deep learning: algorithms and applications*, pages 299–334. Springer, 2020.

[17] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[18] Lilian Weng. Object detection for dummies part 2: Cnn, dpm and overfeat, Dec 2017.

[19] Student notes: Convolutional neural networks (cnn) introduction, Apr 2018. https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/.

[20] Austin. max-pooling, Jan 2019. https://austingwalters.com/convolutional-neural-networks-cnn-to-classify-sentences/.

[21] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[22] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[23] Marco Venturelli. The dangers behind image resizing, Aug 2021. https://blog.zuru.tech/machine-learning/2021/08/09/the-dangers-behind-image-resizing.

[24] Bhavya Rema Devi. Tips for training a 3d-unet model for segmentation tasks., Jun 2021. https://bhavya-rema.medium.com/tips-for-training-a-3d-unet-model-for-segmentation-tasks-5232851d0116.

[25] Mirza Cilimkovic. Neural networks and back propagation algorithm. *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin*, 15(1), 2015.

[26] Roland W Fleming, Ron O Dror, and Edward H Adelson. Real-world illumination and the perception of surface reflectance properties. *Journal of vision*, 3(5):3–3, 2003.

[27] Roland W Fleming, Ron O Dror, and Edward H Adelson. How do humans determine reflectance properties under unknown illumination? 2001.

[28] Peter Vangorp, Jurgen Laurijssen, and Philip Dutré. The influence of shape on the perception of material reflectance. In *ACM SIGGRAPH 2007 papers*, pages 77–es. 2007.

[29] Vlastimil Havran, Jiri Filip, and Karol Myszkowski. Perceptually motivated brdf comparison using single image. In *Computer graphics forum*, volume 35, pages 1–12. Wiley Online Library, 2016.

[30] Yun-Xian Ho, Michael S Landy, and Laurence T Maloney. How direction of illumination affects visually perceived surface roughness. *Journal of vision*, 6(5):8–8, 2006.

[31] Jaroslav Křivánek, James A Ferwerda, and Kavita Bala. Effects of global illumination approximations on material appearance. *ACM Transactions on Graphics (TOG)*, 29(4):1–10, 2010.

[32] Alice C Chadwick and RW Kentridge. The perception of gloss: A review. *Vision research*, 109:221–235, 2015.

[33] Ruiquan Mao, Manuel Lagunas, Belen Masia, and Diego Gutierrez. The effect of motion on the perception of material appearance. In *ACM Symposium on Applied Perception 2019*, pages 1–9, 2019.

[34] Katja Doerschner, Roland W Fleming, Ozgur Yilmaz, Paul R Schrater, Bruce Hartung, and Daniel Kersten. Visual motion and the perception of surface material. *Current Biology*, 21(23):2010–2016, 2011.

[35] Josh Wills, Sameer Agarwal, David Kriegman, and Serge Belongie. Toward a perceptual space for gloss. *ACM Transactions on graphics (TOG)*, 28(4):1–15, 2009.

[36] Fabio Pellacini, James A Ferwerda, and Donald P Greenberg. Toward a psychophysically-based light reflection model for image synthesis. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 55–64, 2000.

[37] Ioannis Gkioulekas, Bei Xiao, Shuang Zhao, Edward H Adelson, Todd Zickler, and Kavita Bala. Understanding the role of phase function in translucent appearance. *ACM Transactions on graphics (TOG)*, 32(5):1–19, 2013.

[38] Bei Xiao, Shuang Zhao, Ioannis Gkioulekas, Wenyan Bi, and Kavita Bala. Effect of geometric sharpness on translucent material perception. *Journal of vision*, 20(7):10–10, 2020.

[39] Ioannis Gkioulekas, Bruce Walter, Edward H Adelson, Kavita Bala, and Todd Zickler. On the appearance of translucent edges. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 5528–5536, 2015.

[40] Filipp Schmidt, Roland W Fleming, and Matteo Valsecchi. Softness and weight from shape: Material properties inferred from local shape features. *Journal of vision*, 20(6):2–2, 2020.

[41] Müge Cavdan, Knut Drewing, and Katja Doerschner. Materials in action: The look and feel of soft. *bioRxiv*, 2021.

[42] Francesca Di Cicco, Maarten WA Wijntjes, and Sylvia C Pont. Understanding gloss perception through the lens of art: Combining perception, image analysis, and painting recipes of 17th century painted grapes. *Journal of vision*, 19(3):7–7, 2019.

[43] Johanna Delanoy, Ana Serrano, Belen Masia, and Diego Gutierrez. Perception of material appearance: A comparison between painted and rendered images. *Journal of Vision*, 21(5):16–16, 2021.

[44] JL Nieves, L Gomez-Robledo, Yu-Jung Chen, and J Romero. Computing the relevant colors that describe the color palette of paintings. *Applied Optics*, 59(6):1732–1740, 2020.

[45] Juan Luis Nieves, Juan Ojeda, Luis Gómez-Robledo, and Javier Romero. Psychophysical determination of the relevant colours that describe the colour palette of paintings. *Journal of imaging*, 7(4):72, 2021.

[46] Roland W Fleming and Katherine R Storrs. Learning to see stuff. *Current Opinion in Behavioral Sciences*, 30:100–108, 2019.

[47] Katherine R Storrs, Barton L Anderson, and Roland W Fleming. Unsupervised learning predicts human perception and misperception of gloss. *Nature human behaviour*, 5(10):1402–1417, 2021.

[48] Ana Serrano, Diego Gutierrez, Karol Myszkowski, Hans-Peter Seidel, and Belen Masia. An intuitive control space for material appearance. *arXiv preprint arXiv:1806.04950*, 2018.

[49] Manuel Lagunas, Sandra Malpica, Ana Serrano, Elena Garces, Diego Gutierrez, and Belen Masia. A similarity measure for material appearance. *arXiv preprint arXiv:1905.01562*, 2019.

[50] Johanna Delanoy, Manuel Lagunas, Ignacio Galve, Diego Gutierrez, Ana Serrano, Roland Fleming, and Belen Masia. The role of objective and subjective measures in material similarity learning. In *ACM SIGGRAPH 2020 Posters*, pages 1–2. 2020.

[51] Kristin J Dana, Bram Van Ginneken, Shree K Nayar, and Jan J Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions On Graphics (TOG)*, 18(1):1–34, 1999.

[52] Eric Hayman, Barbara Caputo, Mario Fritz, and Jan-Olof Eklundh. On the significance of real-world conditions for material classification. In *European conference on computer vision*, pages 253–266. Springer, 2004.

[53] Lavanya Sharan, Ruth Rosenholtz, and Edward Adelson. Material perception: What can you see in a brief glance? *Journal of Vision*, 9(8):784–784, 2009.

[54] Sean Bell, Paul Upchurch, Noah Snavely, and Kavita Bala. Opensurfaces: A richly annotated catalog of surface appearance. *ACM Transactions on graphics (TOG)*, 32(4):1–17, 2013.

[55] Lutz Kettner, Matthias Raab, Daniel Seibert, Jan Jordan, and Alexander Keller. The material definition language. In *Proceedings of the Third Workshop on Material Appearance Modeling: Issues and Acquisition*, pages 1–4, 2015.

[56] Wojciech Matusik. *A data-driven reflectance model*. PhD thesis, Massachusetts Institute of Technology, 2003.

[57] Jirí Filip and Radomír Vávra. Template-based sampling of anisotropic brdfs. In *Computer Graphics Forum*, volume 33, pages 91–99. Wiley Online Library, 2014.

[58] Stephen Lombardi and Ko Nishino. Reflectance and natural illumination from a single image. In *European Conference on Computer Vision*, pages 582–595. Springer, 2012.

[59] Jonathan Dupuy and Wenzel Jakob. An adaptive parameterization for efficient material acquisition and rendering. *ACM Transactions on graphics (TOG)*, 37(6):1–14, 2018.

[60] James A Ferwerda, Fabio Pellacini, and Donald P Greenberg. Psychophysically based model of surface gloss perception. In *Human vision and electronic imaging vi*, volume 4299, pages 291–301. International Society for Optics and Photonics, 2001.

[61] Pascal Barla, Romain Pacanowski, and Peter Vangorp. A composite brdf model for hazy gloss. In *Computer Graphics Forum*, volume 37, pages 55–66. Wiley Online Library, 2018.

[62] Manuel Laguna. *Learning visual appearance: perception, modeling and editing*. PhD thesis, Universidad de Zaragoza, 2021.

[63] Roland W Fleming, Christiane Wiebel, and Karl Gegenfurtner. Perceptual qualities and material classes. *Journal of vision*, 13(8):9–9, 2013.

[64] William Thompson, Roland Fleming, Sarah Creem-Regehr, and Jeanine Kelly Stefanucci. *Visual perception from a computer graphics perspective*. CRC press, 2011.

[65] Tiancheng Sun, Henrik Wann Jensen, and Ravi Ramamoorthi. Connecting measured brdfs to analytic brdfs by data-driven diffuse-specular separation. *ACM Transactions on Graphics (TOG)*, 37(6):1–15, 2018.

[66] Jason Lawrence, Aner Ben-Artzi, Christopher DeCoro, Wojciech Matusik, Hanspeter Pfister, Ravi Ramamoorthi, and Szymon Rusinkiewicz. Inverse shade trees for non-parametric material representation and editing. *ACM Transactions on Graphics (TOG)*, 25(3):735–745, 2006.

[67] Károly Zsolnai-Fehér, Peter Wonka, and Michael Wimmer. Photorealistic material editing through direct image manipulation. In *Computer Graphics Forum*, volume 39, pages 107–120. Wiley Online Library, 2020.

[68] Marlon Mylo, Martin Giesel, Qasim Zaidi, Matthias Hullin, and Reinhard Klein. Appearance bending: A perceptual editing paradigm for data-driven material models. In *Proceedings of the conference on Vision, Modeling and Visualization*, pages 9–16, 2017.

[69] Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. State of the art in artistic editing of appearance, lighting and material. In *Computer Graphics Forum*, volume 35, pages 216–233. Wiley Online Library, 2016.

[70] Manuel Lagunas, Ana Serrano, Diego Gutierrez, and Belen Masia. The joint role of geometry and illumination on material recognition. *Journal of Vision*, 21(2):2–2, 2021.

[71] Jonathan T Barron and Jitendra Malik. Shape, illumination, and reflectance from shading. *IEEE transactions on pattern analysis and machine intelligence*, 37(8):1670–1687, 2014.

[72] Ye Yu and William AP Smith. Inverserendernet: Learning single image inverse rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3155–3164, 2019.

[73] Tom Haber, Christian Fuchs, Philippe Bekaer, Hans-Peter Seidel, Michael Goesele, and Hendrik PA Lensch. Relighting objects from image collections. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 627–634. IEEE, 2009.

[74] Elena Garces, Adolfo Munoz, Jorge Lopez-Moreno, and Diego Gutierrez. Intrinsic images by clustering. In *Computer graphics forum*, volume 31, pages 1415–1424. Wiley Online Library, 2012.

[75] Taishi Ono, Hiroyuki Kubo, Kenichiro Tanaka, Takuya Funatomi, and Yasuhiro Mukaigawa. Practical brdf reconstruction using reliable geometric regions from multi-view stereo. *Computational Visual Media*, 5(4):325–336, 2019.

[76] Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. Material editing using a physically based rendering network. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[77] Konstantinos Rematas, Tobias Ritschel, Mario Fritz, Efstratios Gavves, and Tinne Tuytelaars. Deep reflectance maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[78] Maxim Maximov, Laura Leal-Taixe, Mario Fritz, and Tobias Ritschel. Deep appearance maps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[79] Chloe LeGendre, Wan-Chun Ma, Graham Fyffe, John Flynn, Laurent Charbonnel, Jay Busch, and Paul Debevec. Deeplight: Learning illumination for unconstrained mobile mixed reality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[80] DUAN GAO, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. Deep inverse rendering for high-resolution svbrdf estimation from an arbitrary number of images. *ACM Trans. Graph.*, 38(4), jul 2019.

[81] B Mildenhall, P Srinivasan, M Tancik, JT Barron, and RRR Ng. Representing scenes as neural radiance fields for view synthesis. In *Proc. of European Conference on Computer Vision, Virtual*, 2020.

[82] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik P.A. Lensch. Nerd: Neural reflectance decomposition from image collections. In *IEEE International Conference on Computer Vision (ICCV)*, 2021.

[83] Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *ACM Transactions on Graphics (TOG)*, 40(6):1–18, 2021.

[84] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *CVPR*, 2021.

[85] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797, 2018.

[86] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

[87] Takuhiro Kaneko, Hirokazu Kameoka, Kou Tanaka, and Nobukatsu Hojo. Cyclegan-vc2: Improved cyclegan-based non-parallel voice conversion. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6820–6824. IEEE, 2019.

[88] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[89] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[90] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[91] Wenzel Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[92] Hdrihaven. https://www.hdrihaven.com/.

[93] Kalideo remove.bg. https://www.remove.bg.

[94] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[95] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[96] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

[97] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[98] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[99] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[100] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[101] Jorge Condor Lacambra, Manuel Lagunas Arto, Johanna Delanoy, Belén Masiá Corcoy, Diego Gutiérrez, et al. Normal map estimation in the wild. *Jornada de Jóvenes Investigadores del I3A*, 9, 2021.

[102] Mark D Fairchild and Garrett M Johnson. icam framework for image appearance, differences, and quality. *Journal of Electronic Imaging*, 13(1):126–138, 2004.

[103] Svetlana Grečova and Samuel Morillas. Perceptual similarity between color images using fuzzy metrics. *Journal of Visual Communication and Image Representation*, 34:230–235, 2016.

[104] Cg basics. https://www.racoon-artworks.de/cgbasics/raytracing.php.

[105] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[106] Yani Andrew Ioannou. *Structural priors in deep neural networks*. PhD thesis, University of Cambridge, 2018.

[107] Saul Dobilas. Convolutional neural networks explained-how to successfully classify images in python, Jun 2022.

[108] Mohammad Hemmat Esfe, S Ali Eftekhari, Maboud Hekmatifar, and Davood Toghraie. A well-trained artificial neural network for predicting the rheological behavior of mwcnt–al2o3 (30–70%)/oil sae40 hybrid nanofluid. *Scientific Reports*, 11(1):1–11, 2021.

[109] Autoencoders. https://sci2lab.github.io/ml_tutorial/autoencoder/.

[110] Jeffry S Nimeroff, Eero Simoncelli, and Julie Dorsey. Efficient re-rendering of naturally illuminated environments. In *Photorealistic Rendering Techniques*, pages 373–388. Springer, 1995.

[111] Carlo H Séquin and Eliot K Smyrl. Parameterized ray-tracing. *ACM SIGGRAPH Computer Graphics*, 23(3):307–314, 1989.

[112] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 197–206, 1990.

[113] Aner Ben-Artzi, Kevin Egan, Frédo Durand, and Ravi Ramamoorthi. A precomputed polynomial representation for interactive brdf editing with global illumination. *ACM Transactions on Graphics (TOG)*, 27(2):1–13, 2008.

[114] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *CoRR*, abs/2102.12092, 2021.

[115] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[116] Ana Serrano, Bin Chen, Chao Wang, Michal Piovarči, Hans-Peter Seidel, Piotr Didyk, and Karol Myszkowski. The effect of shape and illumination on material perception: model and applications. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021.

[117] Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák. Scalable realistic rendering with many-light methods. In *Computer Graphics Forum*, volume 33, pages 88–104. Wiley Online Library, 2014.

[118] Ying Song, Xin Tong, Fabio Pellacini, and Pieter Peers. Subedit: A representation for editing measured heterogeneous subsurface scattering. *ACM Trans. Graph.*, 28(3), jul 2009.

[119] Fabio Pellacini. Envylight: An interface for editing natural illumination. In *ACM SIGGRAPH 2010 papers*, pages 1–8. 2010.

[120] Thorsten-Walther Schmidt, Jan Novak, Johannes Meng, Anton S Kaplanyan, Tim Reiner, Derek Nowrouzezahrai, and Carsten Dachsbacher. Path-space manipulation of physically-based light transport. *ACM Transactions On Graphics (TOG)*, 32(4):1–11, 2013.

[121] Mark Christopher Colbert. *Appearance-driven material design*. University of Central Florida, 2008.

[122] Su Xuey, Jiaping Wang, Xin Tong, Qionghai Dai, and Baining Guo. Image-based material weathering. In *Computer Graphics Forum*, volume 27, pages 617–626. Wiley Online Library, 2008.

[123] Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. Deep inverse rendering for high-resolution svbrdf estimation from an arbitrary number of images. *ACM Trans. Graph.*, 38(4):134–1, 2019.

[124] Erum Arif Khan, Erik Reinhard, Roland W Fleming, and Heinrich H Bülthoff. Image-based material editing. *ACM Transactions on Graphics (TOG)*, 25(3):654–663, 2006.

[125] Xiaobo An and Fabio Pellacini. Appprop: all-pairs appearance-space edit propagation. In *ACM SIGGRAPH 2008 papers*, pages 1–9. 2008.

[126] Ivaylo Boyadzhiev, Kavita Bala, Sylvain Paris, and Edward Adelson. Band-sifting decomposition for image-based material editing. *ACM Transactions on Graphics (TOG)*, 34(5):1–16, 2015.

[127] Chuong H Nguyen, Daniel Scherzer, Tobias Ritschel, and Hans-Peter Seidel. Material editing in complex scenes by surface light field manipulation and reflectance optimization. In *Computer Graphics Forum*, volume 32, pages 185–194. Wiley Online Library, 2013.

[128] Shida Beigpour, Sumit Shekhar, Mohsen Mansouryar, Karol Myszkowski, and Hans-Peter Seidel. Light-field appearance editing based on intrinsic decomposition. *Journal of Perceptual Imaging*, 1(1):10502–1, 2018.

[129] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Annexes

Appendix A

Reconstruction Metrics

Sample	Method / Dataset					
	Delanoy et al. / MI-S	**Our STGAN / LD**	**Our STGAN / MED**	**Our STGAN / AVE**	**Our STGAN / MI** ($\lambda_2 = 10$)	**Our STGAN / MI**($\lambda_2 = 50$)
Sample 1	19.83	27.2	28.41	32.17	30.46	27.688
Sample 2	17.56	25.13	27.19	20.73	28.69	26.294
Sample 3	21.78	26.2	28.8	29.64	32.83	32.740
Sample 4	22.43	31.96	28.96	31	31.81	30.073
Sample 5	19.55	27.94	28.55	28.91	33.10	31.378
Sample 6	23.8	25.47	24.03	24.3	22.75	24.308
Sample 7	20.94	17.86	21.9	19.88	17.64	17.604
Sample 8	18.41	28.82	27.69	29.77	30.20	30.001
Sample 9	20.95	27.36	27.85	25.69	24.36	23.821
Sample 10	20.13	29.14	28.09	29.06	29.69	25.852
Sample 11	21.35	32.47	29.08	33.29	30.34	28.488
Sample 12	20.29	30.32	27.46	32.65	31.09	30.636
Sample 13	17.99	27.86	28.19	26.97	31.78	27.925
Sample 14	21.22	30.22	28.47	31.45	29.36	29.006
Sample 15	17.65	24.87	27.74	28.13	29.39	29.027
Sample 16	25.61	27.42	27.24	26.04	27.81	27.119
Sample 17	24.39	25.02	26.26	24.94	23.93	24.928
Sample 18	14.5	24.75	27.69	29.99	30.87	26.667
Sample 19	20.52	25.91	28.53	29.67	32.45	32.379
Sample 20	18.66	29.03	28.15	30.62	30.55	30.238
Sample 21	16.78	28.13	27.32	29.96	30.44	30.176
Sample 22	11.42	25.49	26.28	29.35	29.19	27.917
Sample 23	20.83	32.98	27.74	30.02	28.77	28.700
Sample 24	14.71	20.95	25.18	22.24	22.68	22.994
Sample 25	17.46	30.39	26.95	30.56	30.96	27.679
Sample 26	12.4	28.9	26.22	30.56	29.44	28.631
Sample 27	18.3	28.25	30.32	34.51	31.46	30.835
Sample 28	15.3	25.21	25.71	24.42	29.62	21.492
Sample 29	19.58	26.04	28.26	27.03	29.74	30.293
Sample 30	12.49	31.23	26.77	28.04	26.42	28.822
Sample 31	17.64	18.96	25.76	22.45	19.83	20.354
Sample 32	13.16	25.77	25.89	28.74	27.51	26.590
Sample 33	22.19	25.05	26.86	24.77	25.48	26.014
Sample 34	18.65	28.52	29.51	28.01	29.51	28.073
Sample 35	17.08	24.04	27.03	22.07	22.33	24.572
Sample 36	21.54	29	27.49	26.72	28.67	27.206
Sample 37	17.12	28.66	26.83	21.65	27.47	27.887
Sample 38	18.13	22.59	25.28	22.75	23.13	24.171
Sample 39	17.59	27.41	28	26.74	30.07	27.774
Sample 40	13.31	22.99	25.74	24.52	26.98	25.954
Sample 41	11.79	25.31	25.36	27.35	28.65	27.125
Sample 42	22.55	29.47	28.36	24.42	28.53	29.708
Sample 43	18.72	21.41	25.43	25.67	28.82	25.629
Sample 44	20.35	21.87	24.9	22.13	23.99	25.355
Sample 45	23.8	26.63	26.02	22.72	27.01	26.243
Sample 46	16.47	26.53	26.38	27.86	27.77	26.450
Sample 47	15.87	29.34	28.8	32.59	31.16	31.064
Sample 48	17.19	26.39	29.43	33.47	33.00	31.477
Sample 49	14.45	22.44	26.5	27.46	29.16	27.639
Sample 50	12.47	26.55	27.74	29.83	27.14	27.395
Sample 51	18.14	31.61	29.04	34.02	30.16	28.303
Sample 52	13.73	28.01	25.77	28.46	27.17	24.856
Sample 53	17.87	24.91	25.9	27.4	29.50	27.988
Average	18.24	26.72	27.15	27.61	28.32	27.388

Table A.1: PSNR of reconstructed images of Figure B.3 by the different trained models.

Sample	Method / Dataset					
	Delanoy et al. / MI-S	**Our STGAN / LD**	**Our STGAN / MED**	**Our STGAN / AVE**	**Our STGAN / MI**($\lambda_2 = 10$)	**Our STGAN / MI** ($\lambda_2 = 50$)
Sample 1	0.838	0.97	0.977	0.984	0.978	0.961
Sample 2	0.753	0.942	0.966	0.922	0.967	0.956
Sample 3	0.892	0.948	0.978	0.97	0.981	0.981
Sample 4	0.91	0.986	0.983	0.985	0.987	0.982
Sample 5	0.887	0.972	0.977	0.974	0.986	0.984
Sample 6	0.941	0.974	0.964	0.952	0.957	0.966
Sample 7	0.932	0.937	0.949	0.947	0.929	0.937
Sample 8	0.757	0.97	0.968	0.975	0.974	0.971
Sample 9	0.891	0.978	0.984	0.964	0.957	0.965
Sample 10	0.9	0.981	0.979	0.979	0.978	0.974
Sample 11	0.903	0.987	0.98	0.989	0.982	0.977
Sample 12	0.802	0.973	0.967	0.982	0.977	0.975
Sample 13	0.914	0.986	0.986	0.98	0.993	0.985
Sample 14	0.879	0.978	0.945	0.965	0.975	0.982
Sample 15	0.802	0.956	0.966	0.97	0.973	0.972
Sample 16	0.956	0.977	0.977	0.963	0.973	0.974
Sample 17	0.927	0.968	0.951	0.963	0.951	0.965
Sample 18	0.725	0.938	0.941	0.976	0.977	0.971
Sample 19	0.853	0.938	0.974	0.971	0.979	0.980
Sample 20	0.844	0.974	0.978	0.985	0.981	0.980
Sample 21	0.72	0.963	0.959	0.973	0.974	0.969
Sample 22	0.644	0.944	0.939	0.971	0.968	0.956
Sample 23	0.926	0.988	0.981	0.984	0.985	0.983
Sample 24	0.78	0.936	0.95	0.92	0.942	0.944
Sample 25	0.831	0.982	0.976	0.984	0.985	0.978
Sample 26	0.672	0.965	0.963	0.977	0.974	0.967
Sample 27	0.898	0.971	0.984	0.99	0.987	0.985
Sample 28	0.86	0.961	0.963	0.957	0.980	0.950
Sample 29	0.817	0.924	0.927	0.936	0.960	0.962
Sample 30	0.742	0.986	0.978	0.984	0.984	0.985
Sample 31	0.926	0.95	0.98	0.961	0.954	0.965
Sample 32	0.627	0.949	0.941	0.96	0.951	0.945
Sample 33	0.904	0.958	0.946	0.953	0.967	0.966
Sample 34	0.819	0.955	0.921	0.965	0.970	0.960
Sample 35	0.821	0.953	0.887	0.949	0.948	0.956
Sample 36	0.926	0.985	0.981	0.974	0.982	0.979
Sample 37	0.817	0.972	0.918	0.932	0.969	0.951
Sample 38	0.927	0.976	0.977	0.964	0.978	0.981
Sample 39	0.759	0.929	0.897	0.946	0.954	0.941
Sample 40	0.676	0.937	0.939	0.948	0.963	0.957
Sample 41	0.616	0.944	0.95	0.964	0.969	0.960
Sample 42	0.857	0.956	0.907	0.935	0.967	0.954
Sample 43	0.815	0.934	0.94	0.96	0.970	0.966
Sample 44	0.818	0.939	0.934	0.919	0.950	0.957
Sample 45	0.947	0.979	0.975	0.949	0.980	0.975
Sample 46	0.757	0.93	0.858	0.955	0.940	0.921
Sample 47	0.823	0.974	0.951	0.986	0.979	0.981
Sample 48	0.835	0.967	0.939	0.985	0.978	0.979
Sample 49	0.74	0.947	0.949	0.958	0.971	0.966
Sample 50	0.779	0.972	0.968	0.976	0.973	0.967
Sample 51	0.91	0.989	0.989	0.992	0.988	0.987
Sample 52	0.755	0.969	0.935	0.973	0.960	0.942
Sample 53	0.746	0.947	0.958	0.966	0.973	0.967
Average	0.826	0.961	0.955	0.964	0.970	0.967

Table A.2: SSIM of reconstructed images of Figure B.3 by the different trained models.

Appendix B

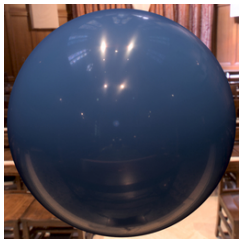
Dataset Images

Sample 1

Sample 2

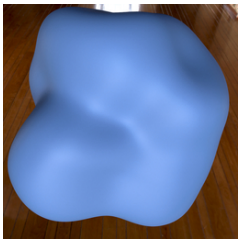
Sample 3

Sample 4



Sample 5

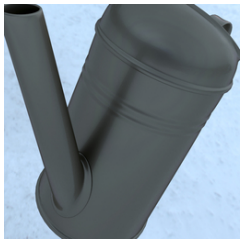
Sample 6



Sample 7

Sample 8

Sample 9

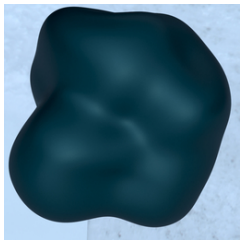


Sample 10

Sample 11

Sample 12

Sample 13



Sample 14

Sample 15

Sample 16

Sample 17

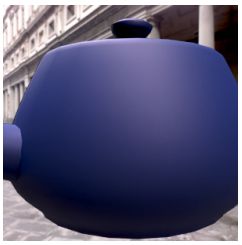
Sample 18

Sample 19

Sample 20

Sample 21

Sample 22



Sample 23

Sample 24

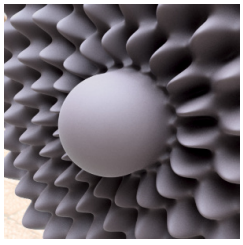
Sample 25

Sample 26

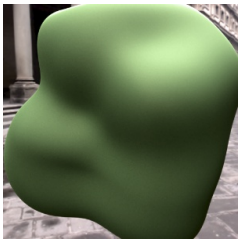
Sample 27

Sample 28

Sample 29

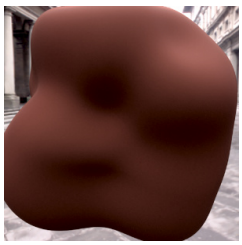


Sample 30



Sample 31

Sample 32



Sample 33

Sample 34

Sample 35

Sample 36

Sample 37

Sample 38

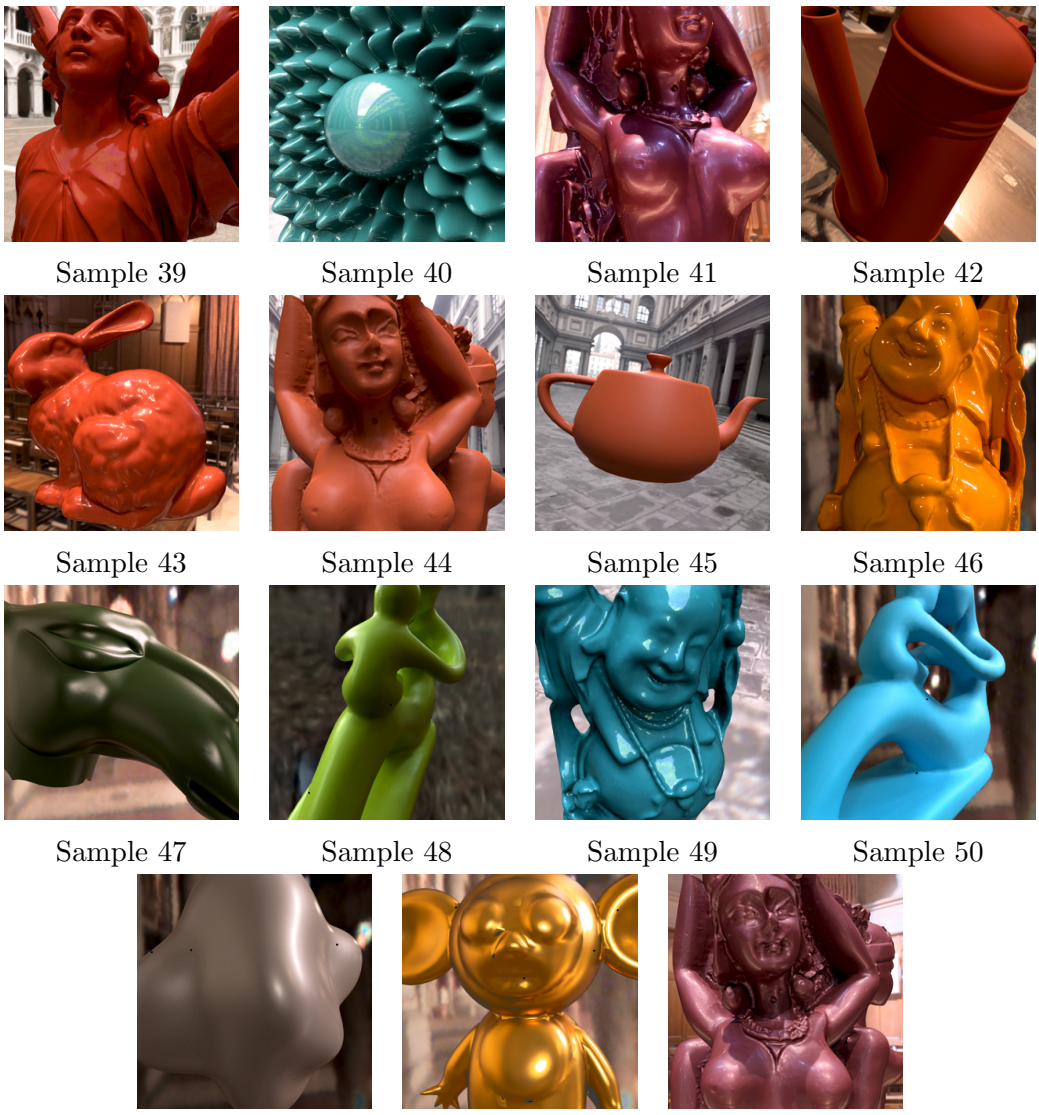


Figure B.3: Synthetic images used to test the reconstruction quality of the models.

Appendix C

Other Results

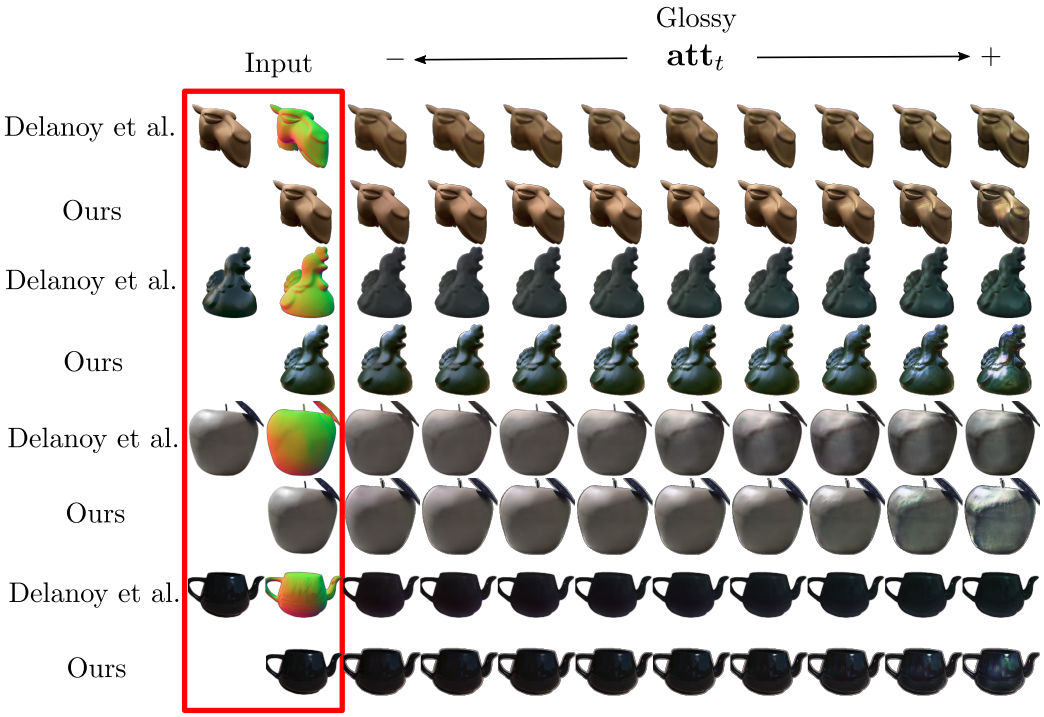


Figure C.1: Each pair of rows are the results generated by our method and the previous work. First column represents the input synthetic images while following ones are the edited images sampling the target perceptual attributes $\mathbf{att}_t$ as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$ for our model while for the other method attributes are sampled as $\{-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1\}$.

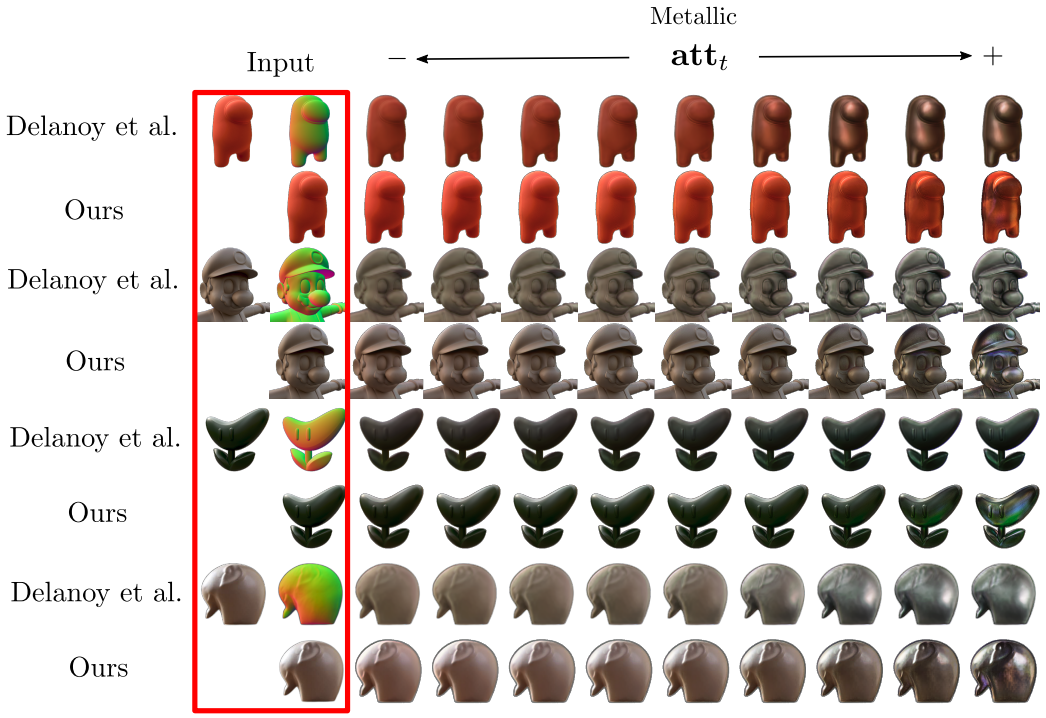


Figure C.2: Each pair of rows are the results generated by our method and the previous work. First column represents the input synthetic images while following ones are the edited images sampling the target perceptual attributes $\mathbf{att}_t$ as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$ for our model while for the other method attributes are sampled as $\{-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1\}$.

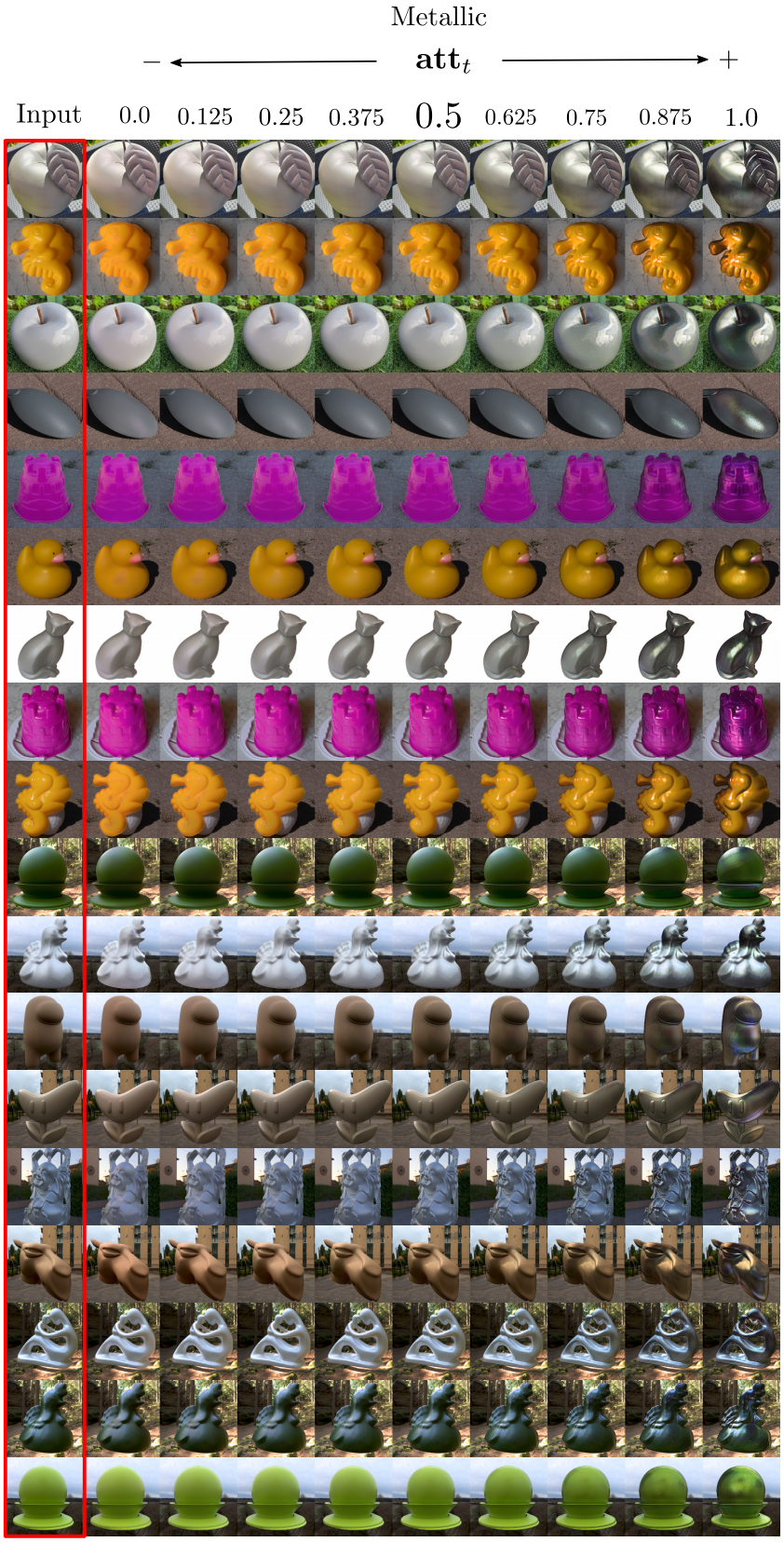


Figure C.3: Edited images varying the target perceptual attribute metallic $\mathbf{att}_t$ sampled as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$ for each input image $\mathbf{x}$.

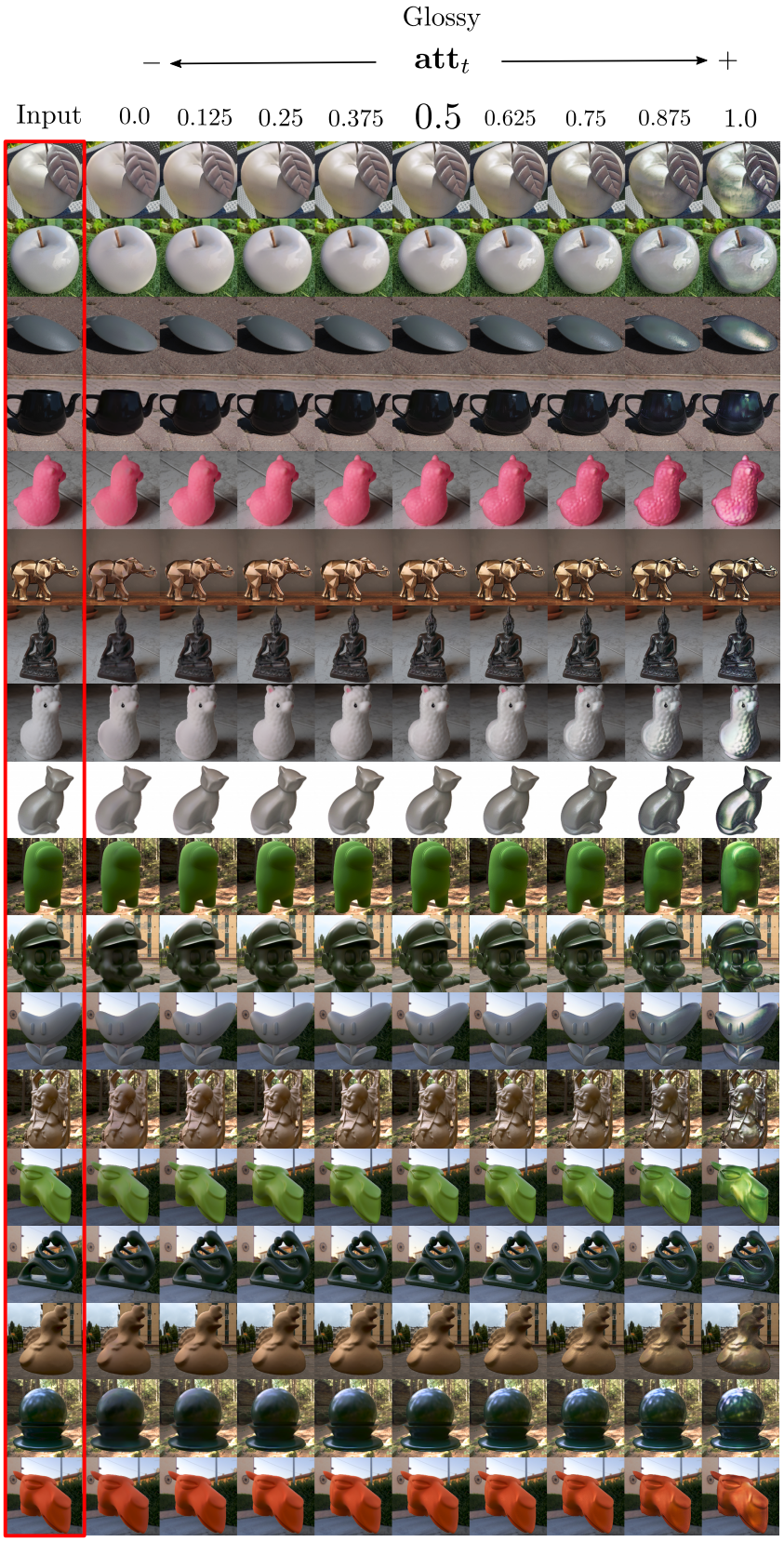


Figure C.4: Edited images varying the target perceptual attribute glossy $\mathbf{att}_t$ sampled as $\{0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$ for each input image $\mathbf{x}$.