UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Computer Systems and Computation

Application of 3D reconstruction techniques for realistic images over drawings and sketches

Master's Thesis

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

AUTHOR: Colom Colom, Joan

Tutor: Martínez Hinarejos, Carlos David

Cotutor: Abad Cerdá, Francisco José

External cotutor: SAITO, HIDEO

ACADEMIC YEAR: 2022/2023

Abstract

Reconstruction from realistic images has evolved very differently when compared to reconstruction from sketches. Even though both present similarities, the latter aims to surpass the subjectivity that drawings present, increasing the task's uncertainty and complexity. In this work, we aim to study the domain of 3D reconstruction from multi-view sketches and drawings by taking inspiration from reconstruction over realistic multi-view images. In contrast to previous reconstruction methods from sketches, we aim to recover not only shape but also color, offering an optimization system that does not require prior training. We make two proposals. Firstly, we present a workflow for applying the state of the art in realistic reconstruction by leveraging NVDiffRec to study its performance over the non-realistic domain. Secondly, we adapt existing methods, using inverse rendering as a refinement process for 3D colored meshes, and propose modifications to work over the domain of drawings. Finally, we highlight the challenges of using both proposals and evaluate how different quality factors in sketches affect the reconstruction quality to determine their viability for fictional 3D content generation from concept art.

Keywords: Computer Vision and Pattern Recognition, Computer Graphics, Image and Video Processing, 3D Reconstruction, Inverse Rendering, Machine Learning.

Resumen

La reconstrucción a partir de imágenes realistas ha evolucionado de forma muy diferente a la reconstrucción a partir de bocetos. Si bien ambos presentan similitudes, el segundo pretende superar la subjetividad de los dibujos, lo que aumenta la incertidumbre y complejidad de la tarea. En este trabajo aspiramos a estudiar el dominio de la reconstrucción 3D a partir de bocetos y dibujos multivista, inspirándonos en la reconstrucción sobre imágenes realistas multivista. A diferencia de los anteriores métodos de reconstrucción a partir de bocetos, este trabajo tiene como objetivo recuperar tanto la forma como el color, ofreciendo un sistema basado en optimización que no requiera entrenamiento previo. Hacemos dos propuestas. En primer lugar, presentamos un flujo de trabajo basado en NVDiffRec para aplicar el estado de la cuestión en reconstrucción realista, estudiando su rendimiento en el dominio no-realista. En segundo lugar, adaptamos métodos existentes, utilizando renderizado inverso para refinar mallas 3D coloreadas y proponiendo modificaciones para aplicarlo en el dominio de los dibujos. Finalmente, exponemos los retos asociados a ambas propuestas y evaluamos cómo diferentes factores de calidad en los bocetos afectan a la calidad de la reconstrucción, determinando su viabilidad para la generación de contenido ficticio 3D a partir de arte conceptual.

Palabras clave: Visión por Computador y Reconocimiento de Formas, Gráficos por Computador, Procesamiento de Imagen y Vídeo, Reconstrucción 3D, Renderizado Inverso, Aprendizaje Automático.

Resum

La reconstrucció a partir d'imatges realistes ha evolucionat de manera molt diferent a la reconstrucció a partir d'esbossos. Si bé tots dos presenten similituds, el segon pretén superar la subjectivitat dels dibuixos, cosa que augmenta la incertesa i complexitat de la tasca. En aquest treball aspirem a estudiar el domini de la reconstrucció 3D a partir d'esbossos i dibuixos multivista, inspirant-nos en la reconstrucció sobre imatges realistes multivista. A diferència dels anteriors mètodes de reconstrucció a partir d'esbossos, aquest treball té com a objectiu recuperar tant la forma com el color, oferint un sistema d'optimització que no necessite entrenament previ. Fem dues propostes. En primer lloc, presentem un flux de treball basat en NVDiffRec per aplicar l'estat de la qüestió en reconstrucció realista, estudiant-ne el rendiment al domini no-realista. En segon lloc, adaptem mètodes existents, utilitzant renderitzat invers per refinar malles 3D acolorides i proposant modificacions per aplicar-lo al domini dels dibuixos. Finalment, exposem els reptes associats a les dues propostes i evaluem com diferents factors de qualitat dels esbossos afecten a la qualitat de la reconstrucció, determinant la seua viabilitat per a la generació de contingut fictici 3D a partir d'art conceptual.

Paraules clau: Visió per Computador i Reconeixement de Formes, Gràfics per Computador, Processament d'Imatge i Vídeo, Reconstrucció 3D, Renderitzat Invers, Aprenentatge Automàtic.

Index

Index of figures

Index of tables

1. Introduction

Since ancient times, sketches and drawings have proven to be powerful mediums for depicting ideas as they convey information visually without words. Therefore they are a powerful communication tool capable of surpassing age, background, or cultural barriers. Indeed, early in life, one of the first skills we learn is to draw and represent our vision of reality on paper.

These are the reasons why, to this day, sketching and developing concept art is a key step in design. From architecture and manufacturing to character and environment design, sketches are a quick and convenient way to settle a concept, its properties, and its appearance [5, 22, 23, 46, 72, 79]. Despite all the advances in human-computer interaction and digital design systems, sketching with pen and paper is the most intuitive medium for professionals to express their ideas.

Throughout this work, we will aim to obtain the 3D information contained in sketches. We will look towards the successful area of multi-view reconstruction from real-life images [20, 33, 44, 52] to draw inspiration. By analyzing their techniques, we intend to apply them to sketches, determining their suitability, advantages, and issues, hoping to open the path for future work.

1.1. Motivation

As stated, sketches and drawings constitute a powerful medium for expressing new and existing content. They are often used as the initial step for designing items in architecture, industrial design, or entertainment industries [5, 23, 79]. To an extent, sketches are flexible and convenient tools to convey 3D shapes without the hassle of dealing with actual three-dimensional matters.

However, the next steps in these design processes often imply formalizing the objects or characters described in the sketches into 3D [5, 72]. This usually involves manually converting these representations into actual 3D models using Computer-Aided Design (CAD) tools [69], which is difficult, requires skill and practice, and offers a less intuitive medium than the original sketches. Therefore, there is a great interest in generating systems capable of automatizing or aiding designers in this process, reducing the cost of 3D model generation and allowing for faster prototyping [25, 41, 42, 46, 75]. Moreover, due to the thriving of augmented and virtual reality and game industries in recent years, the need for 3D content is quickly increasing [16, 46, 74, 76].

On the other hand, the capability of recovering 3D information from 2D content has been highly researched in computer vision since the start [7, 15, 35, 58]. The ability to understand the spatial properties of objects by observation is linked to a better understanding of reality. This not only relates to the capability of recovering 3D information from 2D content but also to a potential improvement in other tasks such as object detection, segmentation, and classification [47, 49].

Sketches are an exciting medium because they can convey 3D information with less visual complexity than real-life images. In contrast, they present a higher ambiguity, and their successful interpretation is highly linked to knowledge, not only involving *recognition* but rather *interpretation*. In this way, the latter requires a deeper understanding of reality and the capability to effectively use this understanding. With all this, the study of systems capable of recovering 3D information from drawings and sketches opens the door to studying this interpretation process, a fundamental step in developing systems with a higher understanding of reality.

Lastly, approaching the task from the more practical side of things, we aim to study how promising results found in reconstruction for realistic images can be applied to the domain of sketches. In this

way, we explore solutions that, to the best of our knowledge, have yet to be applied in the domain of drawings, determining if they are suitable for such a task.

1.2. Objectives

Our objectives with the development of this project are the following:

- Determining if recent techniques successfully developed for 3D reconstruction from realistic images can be applied over the domain of sketches and drawings. We make a particular emphasis on the use of inverse rendering techniques [20, 52].
- Analyzing the particularities of sketches, obtaining a better understanding of the task, its limitations, and requirements. In this way, we want to determine how different quality factors inherent to sketches interfere with 3D reconstruction. By doing so, we aim to establish where future research efforts should be placed.
- Taking the opportunity that this project offers to get not only a better understanding of the state of the art surrounding computer vision and 3D reconstruction but also further practical experience with them. Therefore, we aim to apply, in the range of possibility and reasonability, a wide set of tools and frameworks to cover the needs of our development, increasing our knowledge in the combined field of machine learning and computer graphics.

1.3. Structure

This document presents our research and development through a total of eight chapters. While the current chapter aims to introduce our motivations, intents, and topics, the following chapters will detail the context of our work, its development, and the results we obtained from it. The remainder of this thesis is structured as follows:

- Chapter 2 presents the state of the art about 3D reconstruction. Firstly we introduce previous approaches in reconstruction from sketches. Secondly, we detail current techniques for reconstruction in realistic scenarios and compare both areas to motivate our choices.
- Chapter 3 presents the specific technologies used throughout our work. We focus on the concepts and methods behind differentiable rendering and on providing a detailed overview of the main system used in our first proposal, NVDiffRec [52].
- Chapter 4 details our development, which is structured into two proposals. Firstly, we explain how NVDiffRec can be used over drawings and the associated challenges, suggesting a workflow to be applied when only images are available. Secondly, we leverage techniques from reconstruction over realistic images and adapt them to sketches, building our own system.
- Chapter 5 showcases the results obtained from both proposals, displaying their limitations and capabilities. Moreover, a study is performed on how different quality factors in sketches affect the reconstruction quality.
- Chapter 6 leverages the results to determine if the techniques and systems detailed throughout this work suit our task, obtaining the appropriate conclusions regarding their usability.
- Chapter 7 presents future work for our research, building upon the conclusions obtained.
- Finally, Chapter 8 showcases the references used throughout our work.

2. State of the art

The field of 3D reconstruction from images has been widely covered in the past. In this work, we desire to tackle automatic reconstruction from multi-view sketches. However, for doing so, we will draw inspiration from techniques developed to be applied to photographs or physically based renders, which we will jointly refer to as realistic images.

Consequently, throughout this section, we will first present previous works related to 3D content generation from sketches, covering not only multi-view reconstruction but also single-view reconstruction and interactive approaches, summarized in Table 1. This will help us better understand the state of the art in sketch reconstruction and settle a point of comparison. Next, we will complete this comparison by presenting recent approaches in 3D reconstruction from multi-view realistic images, summarized in Table 2. Both Tables 1 and 2 can be found at the end of this chapter. Finally, we will cover the related area of Structure-from-Motion, which also will be relevant to our development.

2.1. Reconstruction from sketches

The estimation of three-dimensional shapes from sketches has been a broadly researched topic. However, compared with reconstruction from real-life images, it involves additional challenges. Sketches usually lack shading, which can hint at the object's shape, and geometric inconsistencies between different views may appear. Furthermore, there is a lack of sufficient hand-drawn ground truth data paired with 3D models in many cases. Even though some hand-drawn datasets exist, such as [23] and [79], they generally do not provide enough samples for training a system, causing previous works to rely on training with synthetic data [25, 46, 74, 83].

Multi-view reconstruction, from any source, aims to obtain a three-dimensional description of an object depicted in multiple two-dimensional representations through different views. Consequently, both tasks have a similar nature. However, real-life image reconstruction often aims to revert a rendering process, recovering original unknown scene parameters from observations [33, 40, 44, 45, 52, 55]. Therefore, a rendering pipeline is assumed, consistently approximating the laws of light transport, significantly reducing the ambiguity of the task with the number of meaningful references.

In contrast, sketches are not the result of the laws of physics. Instead, they are subjective views of reality [79]. If different people draw the same object, the results will be very different. In the same way, different people interpret sketches differently. This ill-posed nature distinguishes the task over sketches from the task over realistic images. While the latter aims to invert a well-known process and recover the lost scene information, the former aims to overcome subjective interference to build the most plausible object.

In this intent, multiple methods have been developed. Some have presented interactive alternatives involving the user in the reconstruction [41, 42], leaning on the user’s decisions to deal with uncertainty. Other works have tackled automatic reconstruction from single [16, 22, 24, 72, 74, 75, 83] or multi-view sketches [46, 25]. The following sections showcase these approaches.

2.1.1. Interactive approaches

Interactive approaches take advantage of the user's actions to solve the uncertainty. Even though many interactive 3D content generation systems have been developed over the years, we will focus in this section on two recent works closest to sketch reconstruction.

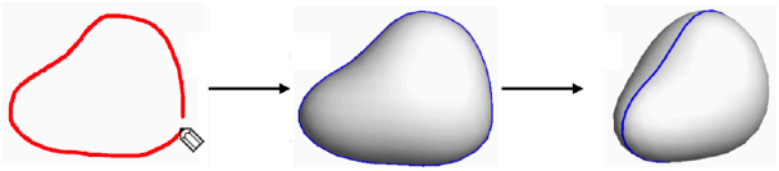

Figure 1. Inflation of a curve [54].

On the one hand, the work by Li et al. [42] exemplified this. In their system, the reconstruction from a reference drawing was guided by the user, who progressively sketched out the regions of the picture to reconstruct, offering a constructive modeling approach. Each traced line was matched against the contours of the image, identifying which parts of the reference should be reconstructed. Additionally, the free form of the sketched line could be maintained if it did not match the reference, allowing for novel shapes. Once a spline was defined for each sketched line, a 3D shape was generated by inflation, as exemplified in Figure 1. In this way, the final 3D model was progressively constructed as a set of inflated splines.

On the other hand, another interactive approach was SketchCNN, proposed in [41]. Through deep learning techniques, they introduced a system capable of generating surface depth and normal estimations from specially styled sketches. SketchCNN was divided into two modules, as seen in Figure 2.

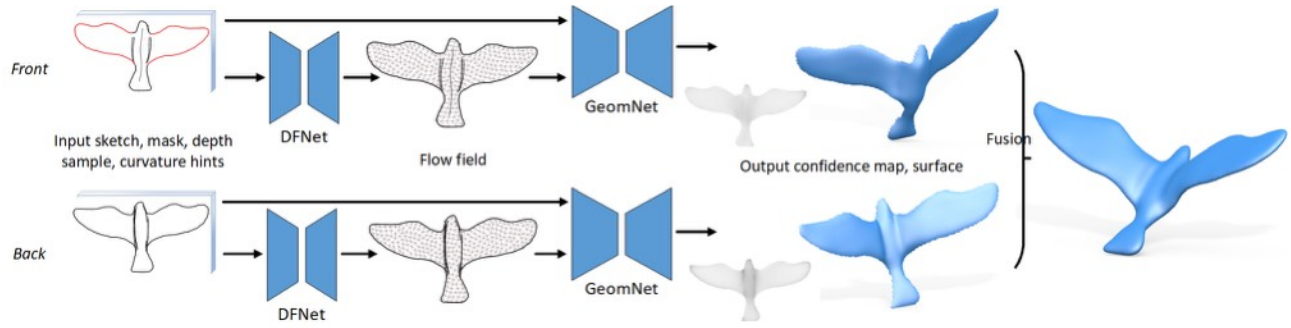

Figure 2. Interactive sketch reconstruction proposal by Li et al. [41].

Firstly, the input maps for the system were the sketch, mask identifying the target, and optional depth and curvature clues. The sketches were encoded following a specific style, in which contour lines were recorded as pure black while other surface lines, such as valleys and ridges, used lower grey levels. From them, a sub-network based on the U-Net architecture [64] was proposed to generate a flow field map, which characterized the curvature of the surface, aiming to help disambiguate the 3D shape. Secondly, all the previous inputs plus the flow field map were fed into a deeper sub-network, which regressed the confidence, normal, and depth maps as result. From them, the 3D surface could be estimated from the corresponding view.

Given the specially annotated inputs, the described system worked automatically until this point. However, a multi-view interactive modeling system was implemented to generate complete shapes. This allowed users to sketch progressively in different views to create closed models by fusing

individual surfaces. Additionally, the interactive approach allowed for incremental sketching, being able to progressively adapt and modify the 3D model to resolve any ambiguities.

2.1.2. Single-view automatic approaches

Another set of works focused on reconstruction from a single view. Wang et al. [74] used a two-module design to generate point clouds from single sketches. First, a standardization module converted sketches to a standard style, being trained to transform distorted synthetic sketches into their original counterparts. Later, the generation module took these standard sketches to obtain viewpoint estimations and generate a point cloud representing the sketched object. The standardization module aimed to solve generalization issues due to training based on synthetic samples, as hand-drawn sketches were able to be converted to a standard common style.

Generating point clouds as well, Gao et al. [16] introduced 3D reconstruction based on sketch translation and a point cloud generator, as seen in Figure 3. The translator followed an encoder-decoder architecture [1], generating spatial features that could extract the 3D information contained in the sketch. The maps generated were processed through convolutional layers to get a density probability map of the sketch, which allowed for sampling 2D points corresponding to the projections of the desired point cloud. Finally, the depth of the points was generated by using the features at the corresponding location. Synthetic samples were used to train the system.

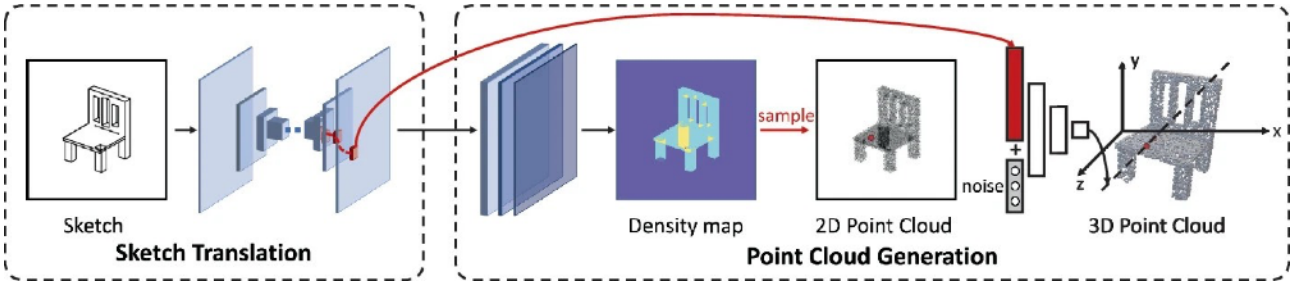

Figure 3. Reconstruction system proposed by Gao et al. [16].

Zhang et al. [83] worked with mesh representations by estimating the deformation of a template mesh through vertex displacements given a single sketch. This was done by explicitly encoding the viewpoint separately from the latent shape vector, allowing for view estimation. To ensure that the viewpoint was considered for the generation, a proposed random view reinforced training combined with adversarial training [21] was used. In contrast to Wang et al. [74], where the network was trained through a Chamfer distance loss [61], this system proposed a silhouette loss by applying differentiable rasterization, shape regularizations, a view prediction loss, and adversarial losses. To generalize to hand-drawn sketches, they proposed an encoder to generate indistinguishable features for synthetic and hand-drawn drawings.

Following a different approach, the work of Wang et al. [75] was based on voxelated representations. To deal with the lack of paired data, they proposed an unsupervised method in which known parings between sketches and models were not required, shown in Figure 4. Instead, an autoencoder [3] was used to project the sketches and the rendered views into a common latent space to obtain similar representations. Then, given the encoding of a sketch, the *N* objects with the nearest render encodings were retrieved. Using another autoencoder for voxel representations, *N* objects were used to generate a final single encoding, which allowed to obtain the final shape. L1 distance with the *N* initial objects and adversarial loss was used to train the shape generator.

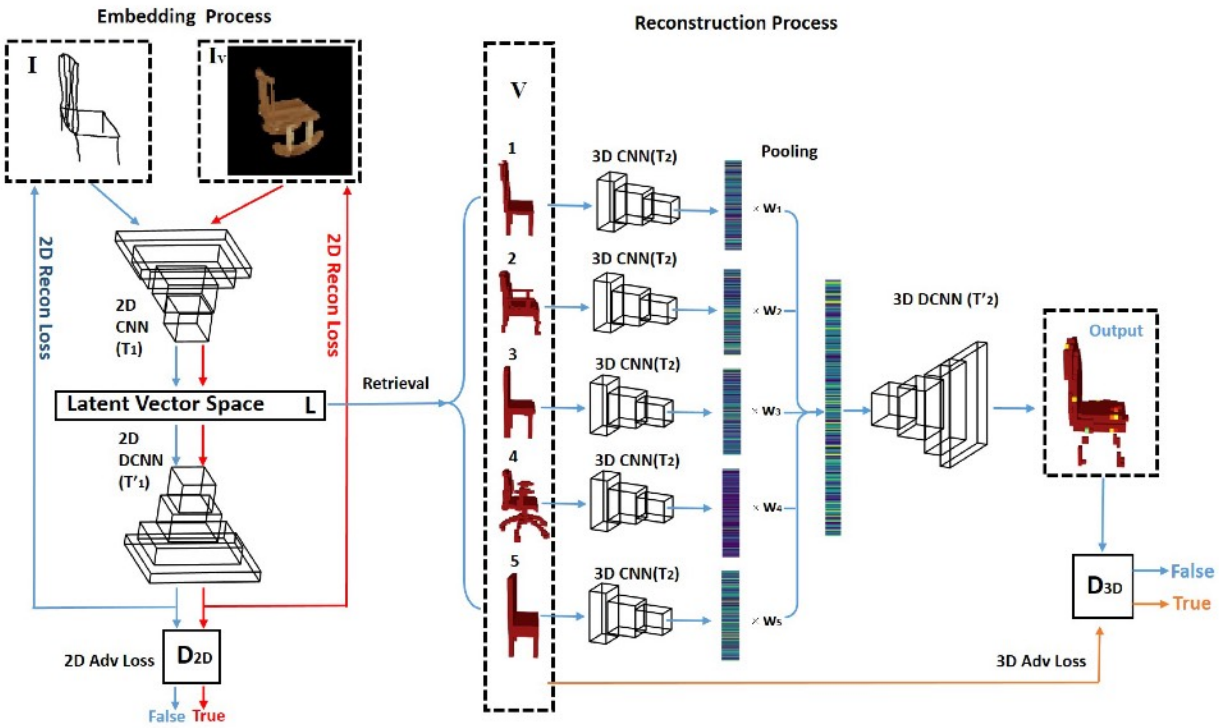

Figure 4. Unsupervised reconstruction system proposed by Wang et al. [75].

So far in this section, the enumerated methods relied on image sketches as input. However, another notable line of work exists aiming to regress 3D information from line sketches encoded as vectorized lines, keeping direct compatibility with tools such as drawing tablets. We will just briefly introduce this line of research as our work focuses on plain image representations.

The approaches over vector strokes generally have a more algorithmic nature rather than using deep learning techniques. Gryaditskaya et al. [22] and Tian et al. [72] tackled the problem by proposing systems capable of obtaining the three-dimensional properties of the intersections found in the 2D sketches. This implied finding the 3D alignment most adequate for the given sketch, driving the reconstruction as the optimization of a score function measuring the quality of the solution. Given the vast range of possible solutions for the problem, both relied on restrictions to make it tractable, simplifying the process and reducing the solution space.

On the one hand, Gryaditskaya et al. assumed that sketches were designed using reference strokes, imposing restrictions regarding connectivity. This allowed them to tackle the problem as a labeling task to classify 2D intersections as *occlusions* or true *intersections*, defining the 3D connectivity. The labeling was done progressively, optimizing a score function capable of measuring how well each possible connectivity fulfilled the established constraints.

On the other hand, Tian et al. applied the parallelism constraint: parallel lines in 2D space must have the same angle with the XY plane. Even though this condition is necessary for 3D parallelism, it is not sufficient, allowing for inaccuracies. This made it possible to tackle the problem of estimating the intersection's depth by grouping lines into classes based on parallelism. In this way, an optimization based on incremental updates of the depths maintaining the defined parallelism was used. This procedure was guided by an optimization function measuring the complexity of the geometry, aiming to minimize it. From the depths, the sketch was able to be converted into a mesh.

Finally, Hähnlein et al. [24] also presented a system based on time-stamped polylines to reconstruct their 3D properties. In this case, a modular sketch design and symmetries were assumed. This

allowed them to divide the sketches into chronological blocks, locating their symmetry planes and determining the symmetry correspondences of each stroke. The method was based on an integer optimization [19] targeting to increase the degree of symmetry and connectivity of the reconstruction.

2.1.3. Multi-view automatic approaches

Despite the wide variety of works tackling reconstruction from single-view sketches, the number of works covering multi-view reconstruction in the same domain is lower. On the one hand, Han et al. [25] worked over a voxel representation and proposed extracting geometric features using a Conditional Generative Adversarial Network (CGAN) [50] to generate attenuation maps for each sketch. This allowed optimizing a voxel grid by a Direct Shape Optimization algorithm [26], obtaining the voxel representation of the target object that best fitted the attenuation maps.

Besides the image sketches, this approach required the view poses too. Even though this system was automatic, they also proposed an interactive editing scheme based on shape retrieval [71] and progressive refining by adding new sketched views. Two alternatives were proposed for dealing with geometric inconsistencies: attributing higher weights to the last sketch or assigning higher weights to the voxels that could be directly inferred. For training, a synthetical dataset was used.

On the other hand, Lun et al. [46] used a similar scheme with different ideas. Through a Convolutional Neural Network (CNN) [38], from sketches, they generated the depth, normal, and foreground probability maps from 12 fixed viewpoints. Using them, partial point clouds were obtained and later fused through optimization to reduce the noisiness and misalignments. Finally, the global cloud was converted into a mesh, which could be further refined through contour fitting.

In this case, the proposed network had to be trained for a fixed set of input views. Therefore, it presented the drawback of assuming a selected set of viewpoints, being a new network retrained for each desired combination of views. As in [25], a synthetically generated dataset was used for training.

2.1.4. Comparison between the approaches

Until now, we have presented a summary of the recent techniques in reconstruction from sketches, introducing the main three approaches. In interactive reconstruction, the user manages ambiguities, allowing for a more precise and satisfactory solution from the user's point of view. Moreover, they offer iterative design, allowing the user to iterate over the reconstruction to improve it. However, this is at the cost of a higher requirement in user effort.

In contrast, automatic approaches imply a lower effort for the user, as the reconstruction is done without their intervention. However, this is generally at the price of a lower capacity of disambiguation. To help solve the issue, automatic deep learning approaches usually rely on intermediate representations capable of extracting the 3D information of the sketches, easing the application of generative methods for point clouds, meshes, or voxels. These systems usually require training and, due to the limitations in paired training data, they use synthetic datasets to learn. This can difficult the generalization to hand-drawn sketches, making it necessary to consider techniques such as standardization or encoding into a shared latent space. Moreover, even when using these techniques, the system will still depend on the types of objects used for training, being difficult to generalize to any kind.

An important area of automatic reconstruction works over vector stroke data. These approaches are algorithmic, not requiring previous training, and being independent of the type of objects drawn. However, they impose restrictions and assumptions on the characteristics of the designs. This, together with the need for specialized inputs, limits their application to specific styles and use cases, more focused on industrial design and architecture.

Finally, multi-view reconstruction has been less explored than single-view. This relates to the difficulty of generating multiple drawn samples for the same object and the inconsistencies that arise between multiple sketches. Despite this, the approaches followed are similar to the deep learning techniques used in single-view. They generate intermediate representations capable of capturing a higher degree of 3D information before generating a 3D shape representation. However, as seen with [46], depending on the approach used, these techniques can be difficult to adapt to a variable number of input views.

Even though it presents a more challenging task, the presence of multiple views has benefits. By observing the target from different points of view, we can better understand the object, helping to disambiguate it. This implies a lower dependence on recognizing the object's class to reconstruct it, being able to have a higher emphasis on *understanding* the shape rather than *recognizing* the shape.

Even though these constitute the main types of approaches, they are not mutually exclusive. Single-view reconstruction could be seen as a particular case of multi-view reconstruction with only one view. Furthermore, [25] and [72] showed that automatic approaches could also be used to build interactive systems, allowing the user to fix any mistakes. This offers a middle ground in which the effort required by the user is lower while offering higher flexibility to obtain the desired output.

2.2. Reconstruction from realistic images

In contrast to reconstruction from multi-view sketches, the generation of 3D representations from multiple realistic views has been broadly studied [10, 20, 27, 35, 48, 52, 67]. Fields such as scene reconstruction, novel view generation, and Structure-from-Motion fall under this category. Although single-view approaches also exist for realistic images, for brevity, this section focuses only on multi-view approaches, as they are the ones we will take as inspiration for our proposals.

As in Section 2.1, we summarize the most recent techniques in 3D reconstruction from realistic multi-view images by presenting the taxonomy of approaches used in the area. First, we cover works using Multi-Layer Perceptrons [18] to represent scenes. Next, we describe works tackling the reconstruction of 3D shapes through explicit representations. Finally, we briefly present the related area of Structure-from-Motion.

2.2.1. Implicit representation

Before covering implicit representation approaches, reviewing the concept of Signed Distance Functions (SDF) –also known as Signed Distance Fields– is important. An SDF establishes the distance of a point in space to an object's surface [11]. Not only that, but the sign of the given distance indicates whether the point is found outside –positive– or inside –negative– of the object. Therefore, the object's surface is characterized by the points of the space whose SDF equals zero. Related to SDFs is the concept of occupancy. In this case, occupancy is a binary function that characterizes the space as full or empty, defining the inside and outside of an object and locating the surface in the frontier [56].

Implicit representations offer an alternative to explicit 3D shape representations such as point clouds, voxels, and meshes. They characterize the 3D properties of the scene through a function, usually modeled as a Multi-Layer Perceptron (MLP) [18] capable of regressing the properties of the space given the position and other factors. Therefore, implicit neural representations can approximate SDFs and occupancy functions to characterize 3D shapes [11, 56, 60, 70].

One of the most notable works in 3D scene reconstruction through implicit representations was NeRF [48]. In this work, scenes were represented through an MLP that returned the emitted color and volume density, given the position in space and the viewing direction, following Figure 5. The position and direction were transformed using positional encodings, allowing for higher detail. Additionally, hierarchical sampling was used thanks to a coarse and a fine network. Initial samples were taken from the coarse network. Then, new samples were extracted from the resulting values near the points with more density, using the fine network to obtain the final values.

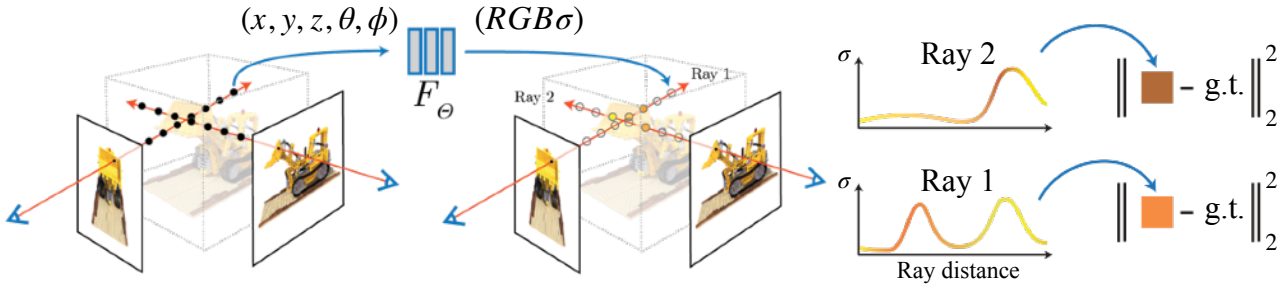

Figure 5. Implicit scene representation for volume rendering proposed by NeRF [48].

Given the position (x, y, z) and viewing direction (θ, ϕ) of a point in space, the color and density σ of said point are returned. This allows the optimization of the MLP through volume rendering.

To ensure consistency, the density was predicted using only the position, as the shape does not depend on the point of view. Once predicted, the viewing direction was appended to obtain the color. The dependency of the color with the viewing direction was necessary to simulate complex view-dependent material effects, such as specular reflections.

With this approach, the reconstruction was modeled as an optimization process, having to train the weights of a new MLP for every new scene. Given the images from multiple points of view and their viewpoints, the scene was optimized through volume rendering, as depicted in Figure 5.

NeRF established a simple but elegant approach that could recover any scene, capturing 3D spaces from their images to offer novel view synthesis and scene inspection. In fact, this approach inspired many subsequent works. Müller et al. [51] proposed an improvement on NeRF based on modifying the encoding used for the inputs. They proposed a trainable multi-level hash table by dividing the space into a multi-resolution voxel grid. In this way, the encodings were generated as a combination of the codes obtained from the voxels containing the point at different resolution levels, enabling efficient high-resolution reconstructions with smaller MLPs.

Also inspired by NeRF, Wang et al. [77] presented NeuS. Focusing on shape estimation, they leveraged SDFs to represent the surface. Therefore, given the position and looking direction, their implicit representation returned the color and the distance of the point to the surface. To learn from images, they adapted the volume rendering equation to work with the SDF values.

Several works tackled the fact that the NeRF scenes could not be relighted as the lighting was baked inside the network. Firstly, Boss et al. [6] presented NeRD. Instead of only learning shape and color, they proposed to learn the shape, Bidirectional Reflectance Distribution Function (BRDF) parameters, and lighting. This way, it was also possible to use source images with different lighting conditions by learning the illumination locally for each image.

The most significant differences of NeRD reside in the estimation of spherical Gaussians for the lighting and the modification of the fine network. While the coarse network estimated the density and a view-independent but illumination-dependent color to guide the sampling, the fine network generated the BRDF parameters and surface normal. By approximating the general rendering equation through a sum of spherical Gaussians, the scene could be rendered from the BRDF and normals. To avoid inconsistencies, the surface and normal estimations were coupled. Meanwhile, to ensure smoothness in the BRDF, its prediction used an autoencoder to map similar BRDFs to the same encoding, improving the surface consistency. Finally, the modifications proposed also allowed the extraction of consistent texture meshes in contrast to NeRF.

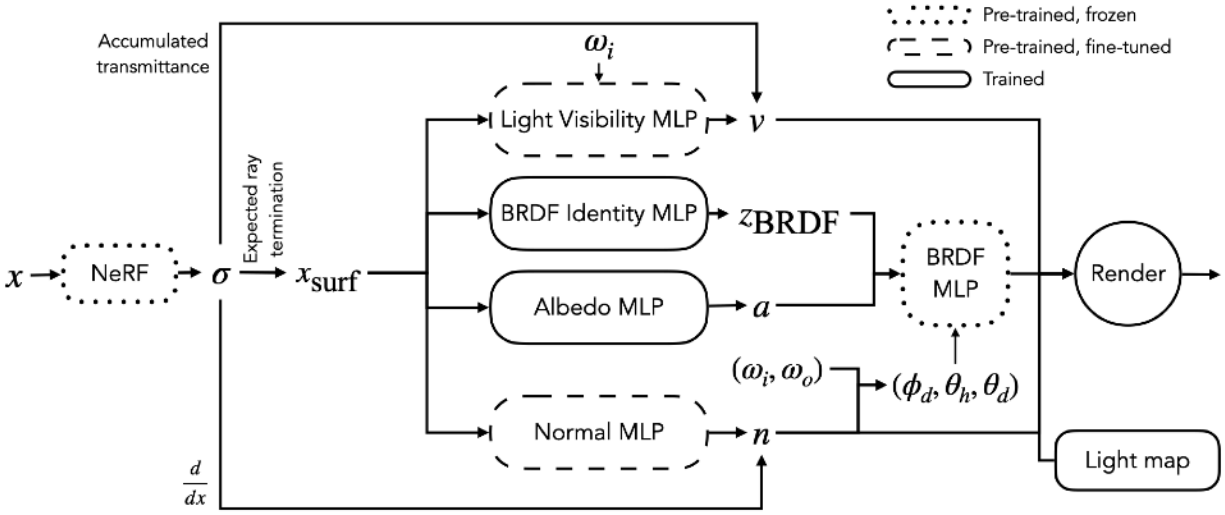

Figure 6. Architecture for implicit reconstruction proposed by NeRFactor [84].

From the initial NeRF estimation, the surface position x_{surf}, normal n, light visibility v, albedo a, and BRDF latent code z_{BRDF} are obtained. x represents a point in space, σ is the density, and ω_i is the light direction, ω_o represents the view direction, and $(\phi_d, \theta_h, \theta_d)$ are Rusinkiewicz coordinates [65].

Secondly, Zhang et al. [84] aimed to provide a re-lightable reconstruction, solving the noise problems derived from the NeRF optimization. This was accomplished through NeRFactor, which leveraged NeRF as an initial step to obtain a base volumetric representation and whose architecture is displayed in Figure 6. Once this representation was initialized, NeRFactor's additional modules could be optimized to refine the estimations and obtain a re-lightable scene.

Concretely, explicit surface points, normals, albedo, specular BRDF, and light visibility were refined from the NeRF scene. The noise was reduced using MLP modules, trained to induce smoothness while maintaining proximity with the direct estimations. A particularity of this system was using a frozen pre-trained decoder for predicting real BRDF values from estimated encodings. Moreover, lighting was optimized directly, learning the values of a High Dynamic Range (HDR) light probe. The optimization was guided by Physically Based Rendering (PBR) equations considering local lighting, optimizing NeRFactor until the initial estimation, excluding it.

Thirdly, PhySG [82] not only intended to obtain re-lightable scenes but also to allow editing materials. With this aim, Zhang et al. proposed encoding the shape through an MLP SDF, as in [77], while modeling environmental lighting with spherical Gaussians, as in [6]. To capture materials, an MLP was used to define a spatially varying albedo based on the surface point, and spherical Gaussians were used to capture the specular BRDF. By sphere tracing and approximating the rendering equation through the spherical Gaussians, the system could be optimized from images, not considering self-occlusion or indirect lighting. An image-based L1 loss and regularizations to enforce non-negative SDF values in the background and unitary normals were used to this end.

Lastly, we close this section with a different type of approach also based on NeRF, proposed by Yang et al. [80]. As input, they used multi-view and multi-light images. Therefore, not only multiple views of the object were provided, but also multiple illuminations for each view. Multi-light images could be used to estimate normal maps, which helped to regularize an initial NeRF estimation. Later, similar to [84], this initialization was refined by optimizing MLPs to obtain refined normals, light visibility, and materials. BRDF materials were encoded through an albedo MLP and a specular MLP, characterizing the latter the weights of spherical Gaussians. Finally, the lighting for each image was determined by a light with a learnable intensity and direction. As in previous works, this system was optimized per scene based on image loss and regularizations.

2.2.2. Explicit representation

As we have seen, implicit representation methods use MLPs as functions to encode shape and material properties. This has advantages like a lower memory footprint, higher flexibility, and a theoretically infinite resolution. However, we have also seen how they present the caveat of difficult access to the optimized content. Many works try to soften this limitation by using SDFs to ease the generation of meshes from the implicit representation, as well as the direct estimation of BRDF materials and lighting maps to be able to edit the scene's materials and illumination.

In contrast, explicit approaches deal with these difficulties by directly optimizing a defined data structure for spatial representation, generally in the form of point clouds, meshes, or voxels. Even though this usually implies an increase in memory footprint with geometry resolution, the explicit representation of the shape and materials allows for easier recovery of the optimized contents. In turn, this facilitates the modification of scenes and their integration with standard systems for 3D content manipulation.

In 2016, Kim et al. [35] proposed a refinement approach for reconstruction based on multi-view images. Like NeRF [48], an optimization scheme was also assumed, refining the shape for each scene given the source images. However, in this case, instead of optimizing a network, the displacement of the mesh's vertices was directly optimized. Starting from an initial Structure-from-Motion estimation [67], the shape and the view poses for each source image were obtained. Once initialized, the mesh, its albedo, and per-camera lighting were optimized using a simple Lambertian without complex reflections as the rendering equation. The rendering error regarding the source images was used to guide the refinement, being normalized per vertex using the number of visible cameras, complemented with geometric and photometric regularizations. While the former aimed to minimize the surface curvature, the latter regularized the ambiguity between albedo and lighting by enforcing that vertices with similar albedo should have a similar color.

This work describes the base approach for the explicit representation reconstruction techniques, which would be further defined in Hasselgren et al. [27]. They built upon the method, using

triangular meshes and PBR-based materials to represent the scene. Through optimization based on differentiable rendering using deferred shading –which will be detailed in Section 3.1.1– and an image loss, they were able to optimize textured shapes. However, they extended the method to also refine aggregated geometry, displacement maps, or skinning based on given animations.

Additionally, they complemented the loss function with a Laplacian regularization, which is widely used in these approaches. As the optimization directly manages the vertices' positions, they can be moved freely without geometry regularization, leading to potentially degenerated stages in which the mesh triangles may clip or overlap. The Laplacian regularization solves this by enforcing geometry smoothness and helping to maintain the relative positions inside vertex neighborhoods.

However, pure mesh representations are not the only ones used in the reconstruction. Mixing ideas from explicit representations and the use of SDFs, Gao et al. [17] and Shen et al. [68] settled for reconstruction based on deformable tetrahedral grid representations that stored SDF or occupancy values at their vertices. In this representation, meshes were optimized explicitly but represented through a gridded space, reducing the problems related to degenerations.

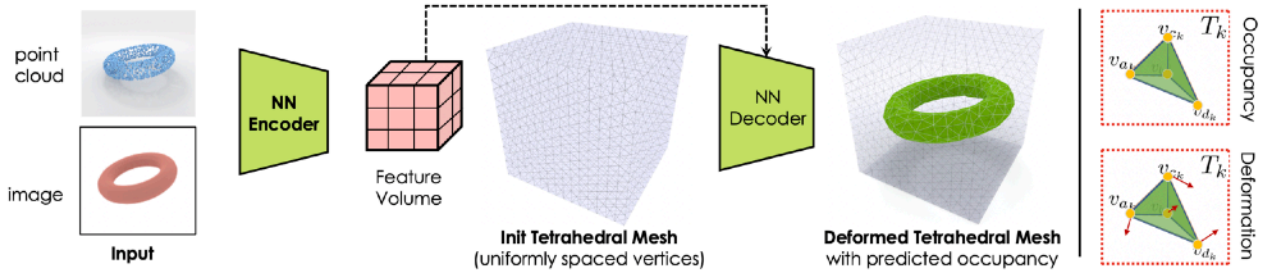

Figure 7. Deformable tetrahedral grid representation proposed by Gao et al. [17].

On the one hand, Gao et al. [17] initially proposed these representations thanks to a middle approach between voxels and meshes, shown in Figure 7. The base was a tetrahedral grid composed of vertices, triangular faces, and tetrahedrons. By characterizing the tetrahedrons with occupancy, the mesh was defined by the faces shared between empty and occupied tetrahedrons. Moreover, to avoid the problems of aliasing, a deformable grid was defined to better adapt to the desired shape.

Gao et al. proposed two possible methods for reconstruction: using gradient-based optimization to refine displacements and occupancies, similarly to [48], or using trained networks to predict the occupancies and vertex displacements, as in Figure 7. This work computed the occupancy per vertex, defining the tetrahedron's value as the maximum of its vertices. Even though this representation circumvented problems related to mesh degeneration, it had to avoid the tetrahedrons flipping.

On the other hand, Shen et al. [68] built over the proposal in [17] to define a system capable of recovering high-resolution models from low-resolution 3D representations. Despite tackling a different task than us, their extension of [17] is relevant to our work. They proposed using SDFs instead of occupancy and a Differentiable Marching Tetrahedra (MT) layer to convert the SDF values into meshes. First, given the surface tetrahedrons identified by different SDF signs in their vertices, the MT algorithm located the topology inside them. Then, the SDF values s were used to compute the positions of the vertices of the resulting triangle faces, as depicted in Figure 8, obtaining the mesh. It is important to note that both [17] and [68] normalized the tetrahedral grid to a unit cube size.

Building upon these ideas, Munkberg et al. [52] presented NVDiffRec. Even though this system will be described in further detail in Section 3.2, we provide a summary here. NVDiffRec proposed

a two-phase optimization over meshes intending to obtain exportable 3D reconstructions ensuring compatibility with standard tools. The deformable SDF tetrahedral grid was used as an internal representation to accomplish this. For recovering materials, an implicit MLP representation was used in the first training phase while converting them to learnable texture maps in the second phase. Finally, the environment light was also optimized thanks to learnable cube maps.

This optimization was possible through differentiable rendering and an image loss, accompanied by regularizations. Therefore, the scene was reconstructed by providing the images, viewpoints, and masks, obtaining a full textured mesh and an HDR environment map. The relevance of NVDiffRec resides in its capability of generating complete scenes with promising results and maintaining compatibility, recovering formats usable by off-the-shelf tools.

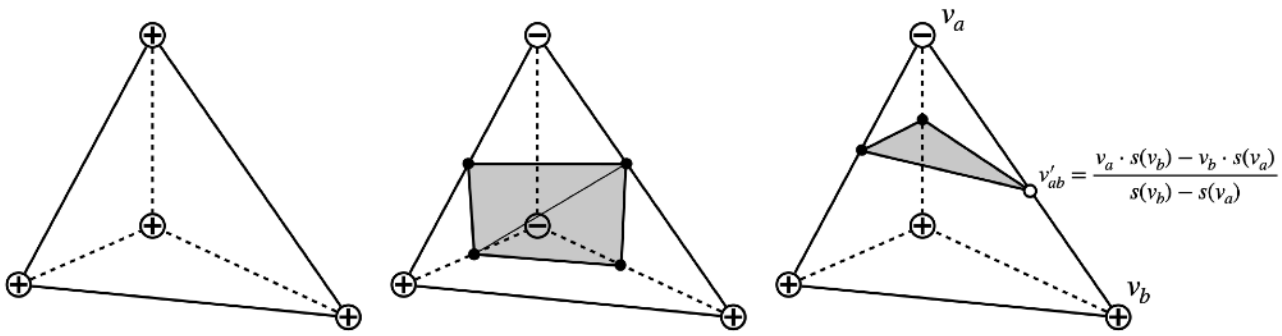

Figure 8. Possible unique topologies inside a tetrahedron following MT [68].

As we have seen so far, most approaches for reconstruction, wherever explicit or implicit, rely on rendering to obtain feedback and guide the optimization of the reconstruction. This rendering is usually tailored to the representation, approximating the general rendering equation in different ways. Nonetheless, looking to balance efficiency and realistic-looking results, these approaches apply simplifications, generally focusing on local lighting without shadows or complex reflections. Even though this allows faster refinements, it is usually a limiting factor in the quality of the reconstructions.

Goel et al. [20] proposed a reconstruction method using differentiable path tracing to avoid the limitations of local lighting, as seen in Figure 9. Starting from an initial mesh estimation, they applied two alternating steps repeated cyclicly: a material refinement step focused on estimating BRDF parameters and a geometry refinement step updating the vertex's positions. After convergence on each cycle, face subdivision was used to increase the resolution of the mesh, followed by simplification to limit the number of parameters and remeshing to fix artifacts.

Several initialization methods were used in [20], such as voxel carving and COLMAP estimations [67], reporting better performance with the second one. Multi-view images, viewpoints, environment maps, material masks, and object masks were given for the reconstruction. For representing the shape, triangle meshes were used, optimizing their vertex positions. A coarse-to-fine approach was applied to avoid falling in local minimums, thanks to a low-res initial mesh and the progressive resolution increase through face subdivision.

For encoding the materials, mesh colors were used [81]: a one-dimensional color vector accessed given the resolution level, triangle index, and barycentric coordinates. This allowed a more straightforward optimization when considering malleable shapes, as learnable image textures would require joint color and texture coordinates optimization. As with shape, the colors were also recovered in a coarse-to-fine fashion. Initially, constant diffuse and specular colors were optimized to avoid baking-in geometric details. After some cycles, spatially varying diffuse color and constant

specularity were refined. Finally, both spatially varying diffuse and specular materials were optimized. To avoid baking specularities into the diffuse color, specular glows were detected and masked.

In this case, the optimization was driven thanks to the Mean Squared Error between renders and ground truth images, not reporting the use of any shape regularization (just a variance penalty for specular colors). As the lack of regularizations could lead to non-recoverable situations, Poisson remeshing after each cycle was used to fix possible issues. It is important to note that after remeshing, the materials were reset to a neutral grey color, effectively discarding the previous color estimations.

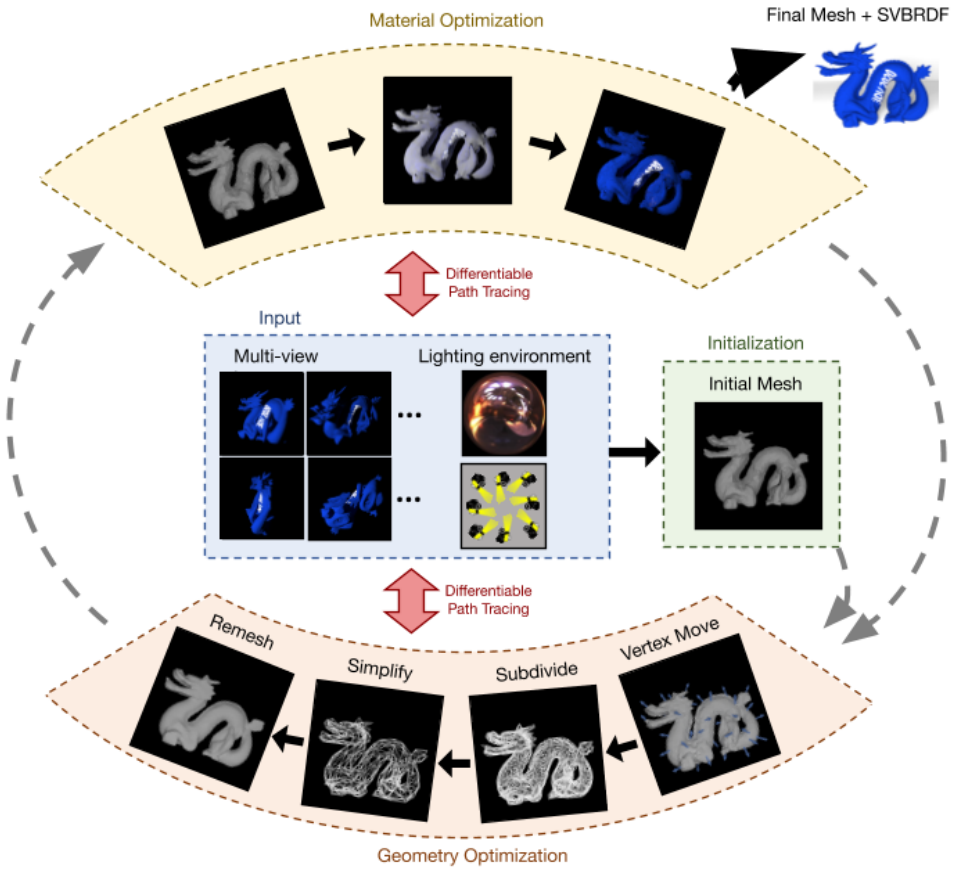

Figure 9. Reconstruction scheme proposed by Goel et al. [20].

2.2.3. Comparison with reconstruction from sketches

When comparing the presented techniques for realistic reconstruction to those from sketches, we can observe how the former often rely on an optimization approach guided by inverse rendering. Meanwhile, non-interactive sketch reconstruction approaches mostly rely on trained deep models to regress 3D information that can be used in the reconstruction. Moreover, while reconstruction from realistic images focuses on joint estimation of both shape and materials, reconstruction from sketches is limited to shape. Even though this is reasonable when considering pure line sketches, we argue that, when dealing with colored sketches or concept art, considering color becomes highly desirable as it is crucial to define the visual identity of an object.

Therefore, when wishing to recover color from sketches for reconstruction, our nearest sources of inspiration lie in reconstruction from realistic images. One advantage of the optimization-based

approaches resides in their higher degree of generality. While deep learning systems are trained under specific object classes and styles, optimization approaches are designed to approximate the best reconstruction possible to the given inputs, reducing the bias.

Nonetheless, optimization methods also present some limitations. They require time to refine a reconstruction for each case, and they are designed with a particular domain in mind. Consequently, the quality of results varies when applied to other fields, even though they are inherently more generic.

2.2.4. Structure-from-Motion

Before closing off the review of the state of the art, discussing the related task of Structure-from-Motion (SfM) [58, 67] is relevant. While reconstruction from multi-view images tries to recover a target or a scene from a given number of images, it generally involves some restrictions or assumptions such as known viewpoints, constant illumination, shared camera parameters, or object masks. In contrast, SfM aims to process extensive image collections of a mutual landmark or scene from multiple sources, such as Internet image collections, to recover the underlying structure. Therefore, SfM deals with a broader task: images of unknown nature and bigger datasets. This more general nature of SfM implies not only recovering the scene structure but also estimating the camera poses. This makes the techniques in this area commonly used as preprocessing steps for reconstruction, whether for obtaining viewpoints or providing an initial shape estimation.

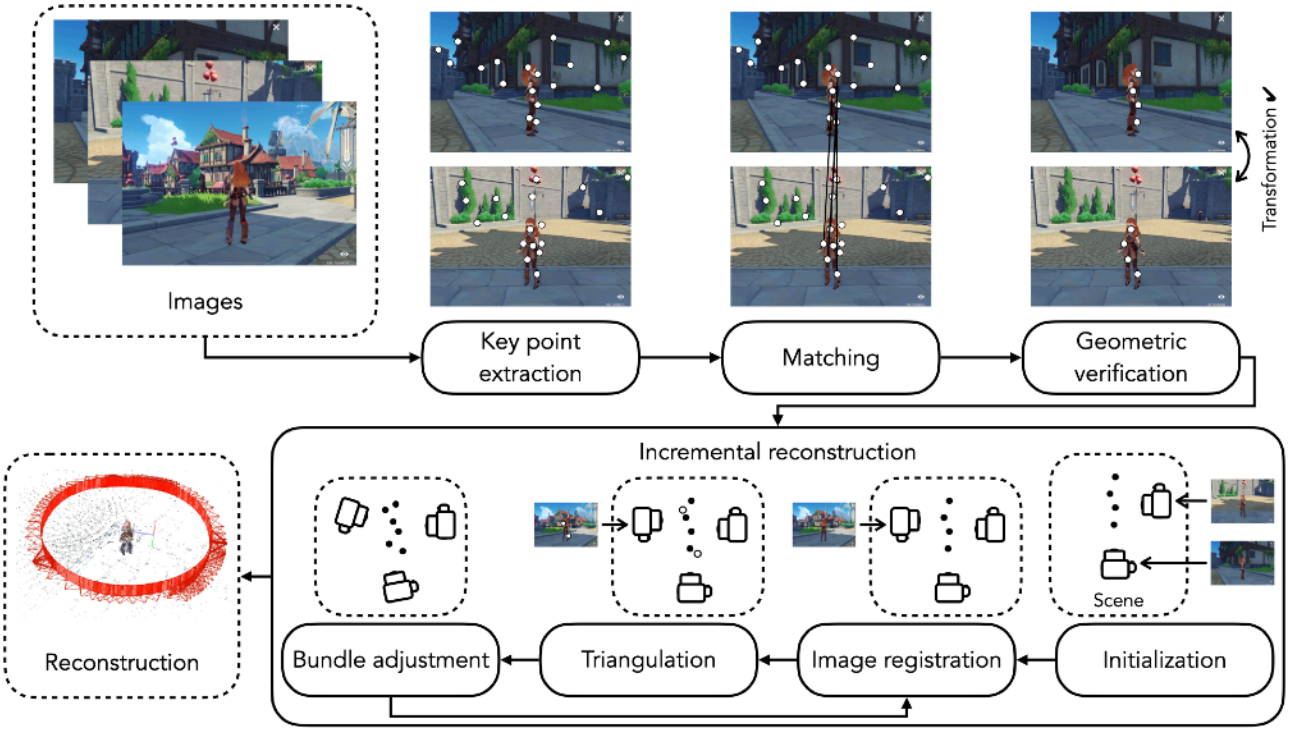

Figure 10. General pipeline for incremental SfM [67].

One of the most notable works in SfM in recent years has been COLMAP [67]. This system, openly provided to the community, was developed as a result of the work of Schönberger et al. and has been widely used since then. Said work provided a general overview of the existing approach for incremental SfM and contributed improvements that helped increase its robustness and efficiency. COLMAP was the result of their improved algorithm.

Following [67], the general pipeline for incremental SfM systems consisted of four steps, shown in Figure 10. Firstly, images were processed to extract characteristic key points. Secondly, the correspondences between key points in different images were identified, forming a list of pairs of potentially overlapping images. Thirdly, the image pairs were geometrically validated to filter the true correspondences. This was done by finding the transformation that mapped a sufficient number of key points between images. Finally, the incremental reconstruction step was applied, progressively generating the view poses and the set of 3D points characterizing the scene.

The incremental reconstruction started from a two-view reconstruction process. This first step was critical as a bad initialization could lead to unrecoverable states. Then, images were progressively added using the correspondences with triangulated points from already registered images, estimating their pose and intrinsic parameters. In turn, registering more images contributed to more points consistently viewed and their triangulation. Finally, bundle adjustment jointly refined both camera and point parameters.

Building upon this basic pipeline, Schönberger et al. [67] introduced improvements to increase the accuracy and robustness of the reconstruction. Firstly, they augmented the image correspondences (also known as scene graph) with additional information to find a robust initialization more efficiently. Specifically, by computing the fundamental, homography, and essential matrices, the number of inlier key points complying with these matrices was added. Watermarks, timestamps, and frames were also detected and discarded as key points.

Secondly, a next-best view selection scheme was proposed for efficiently dealing with extensive collections. In this scheme, images were scored considering the number of triangulated points seen –the higher, the better– and the uniformity of their distribution –the more uniform, the better–. The latter was measured by dividing the images into multi-resolution grids and adding the weight of the cells containing key points at each level, favoring a uniform distribution.

Thirdly, a robust and efficient triangulation system for dealing with outlier contamination was proposed. With this aim, initially, a set of pairs of image observations and poses was said to contain an unknown number of inliers. Then, iteratively, the well-conditioned features were identified based on sufficient triangulation angle, positive depths, and reprojection error lower than a threshold.

Fourthly, the bundle adjustment scheme was modified to mitigate accumulative errors. Local adjustment of the most connected images was applied after each registration. Conversely, global adjustment was only used after reaching a certain reconstruction size. Filtering was applied to delete observations with significant reprojection errors and degenerate cameras. Re-triangulation was also used both before and after the global bundle adjustments for better estimation. Lastly, these steps were applied iteratively until the number of filtered observations and points decreased.

Finally, the number of views was reduced to alleviate the bottleneck caused by bundle adjustment. First, images and points were divided into two groups, depending on whether they were affected by the last incremental step. Considering that bundle adjustment naturally affects more the newly added parts, the images non-affected were grouped in N groups of highly overlapping cameras, fusing each group into a single camera. Meanwhile, the remaining images were maintained as individuals to allow a better refinement.

Thanks to the improvements introduced in [67], the more efficient and robust reconstructions, and the easy-to-use access provided to its implementation, the use of COLMAP has been widespread, becoming a reference and staple in SfM. Nonetheless, other approaches have been proposed since then, such as the works of Cui et al. [10] and Kong et al. [37].

On the one hand, Cui et al. [10] proposed a hybrid approach to combine the advantages of both incremental and global SfM. In contrast to the incremental process of COLMAP, global techniques relied on a single bundle adjustment after estimating simultaneously all camera poses. This was possible thanks to a rotation and translation averaging step. While the former estimated all the camera rotations from relative orientations between overlapping images, the latter tried to estimate the camera positions. Compared to incremental approaches, global techniques avoided the need for a seed model and possible bad initializations, reducing the use of bundle adjustment.

However, they also presented some drawbacks, as they were more sensitive to the errors and outliers in the initial scene graph estimation. Therefore, [10] proposed a hybrid approach to improve the cost of incremental methods and the robustness of global techniques, striking a good balance between quality and efficiency.

The proposal of Cui et al. was divided into two stages. Firstly, an initial rotation estimation inherited from the global approaches was defined. Secondly, inheriting from incremental techniques, center estimation was used. The initial pair of cameras was chosen based on the higher number of matches, wider angle, and higher number of connections. From the initialization, camera registration, triangulation, and bundle adjustment were applied iteratively, keeping constant the intrinsic parameters and rotations in the latter.

On the other hand, the work of Kong et al. [37] dealt with 3D shape and pose estimation given 2D images with annotated landmarks of a given object class. This was approached through Non-Rigid SfM, considering the objects of the same class as deformations of a representative class object. To model a sparse solution space characterizing deformations as combinations of smaller steps, an encoder-decoder was proposed to learn hierarchical dictionaries and sparse encodings. This allowed obtaining the viewpoint and 3D coordinates for the landmarks, recovering the object's structure.

As a summary to close this chapter, Tables 1 and 2 present an overview of Sections 2.1 and 2.2. Both tables follow the same structure. For each paper presented, the number of views required by their proposed system is indicated as 1 (single-view) or N (multi-view). Moreover, the "Automatic" and "Interactive" columns indicate if the described approach was capable of performing complete reconstruction automatically and if an interactive strategy involving the user was presented, respectively. Next, a brief compilation of the keywords describing the main technologies used on these systems is shown. Finally, the last columns indicate the inputs required by the systems to apply reconstruction (once trained, if needed) and their outputs by indicating in which form the shape, materials, and lighting of the target were recovered.

Paper	Number of views	Automatic	Interactive	Technologies	Input	Shape	Materials	Lighting
Li et al. [42]	1	✗	✓	Algorithmic. Contour matching, splines, curve inflation.	Image sketches. Interactive user sketching over the reference.	Meshes	✗	✗
Li et al. [41] (SketchCNN)	N	✗	✓	Deep learning (generative) + Algorithmic. U-Nets, flow field maps, depth and normal maps. Reconstruction losses and regularizations, fusion optimization.	Specially styled image sketches, masks, optional depths, optional curvature.	Meshes	✗	✗
Wang et al. [74]	1	✓	✗	Deep learning (generative). Sketch standardization, view estimation, generative networks. Chamfer distance loss and regularization.	Gray-scale image sketches.	Point clouds	✗	✗
Gao et al. [16] (SketchSampler)	1	✓	✗	Deep learning (generative). Sketch translation, encoder-decoder, Convolutional Networks, Multi-Layer Perceptron. Reconstruction losses.	Image sketches.	Point clouds	✗	✗
Zhang et al. [83] (Sketch2Model)	1	✓	✗	Deep learning (generative). Explicit disjoint view and shape encoding, encoder-decoder, common features for hand-drawn and synthetic sketches, differentiable rendering. Silhouette loss, view loss, adversarial losses and regularizations.	Image sketches.	Meshes	✗	✗
Wang et al. [75]	1	✓	✗	Deep learning (generative). Unsupervised, autoencoder, common features for hand-drawn sketches and renders, shape retrieval. Reconstruction and adversarial losses.	Image sketches, database of 3D models.	Voxels	✗	✗
Gryaditskaya et al. [22]	1	✓	✗	Algorithmic. Intersection labeling for 3D connectivity estimation, optimization. Score function based on constraints.	Polylines with timestamps representing sketches with reference lines.	3D lines	✗	✗
Tian et al. [72]	1	✓	✓	Algorithmic. Intersection depth estimation, parallelism constraint, optimization. Score function based on complexity.	Vector strokes (2D graph).	Meshes	✗	✗
Hähnlein et al. [24]	1	✓	✗	Algorithmic. 3D symmetries estimation, integer optimization. Score function measuring the degree of symmetry and connectivity.	Polylines with timestamps representing sketches with symmetries.	3D lines	✗	✗
Han et al. [25]	N	✓	✓	Deep learning (generative) + Algorithmic. Conditional GAN, attenuation maps, Direct Shape Optimization, differentiable attenuance rendering. Image loss, reconstruction loss and adversarial loss.	Image sketches, viewpoints, optional database of 3D models.	Voxels	✗	✗
Lun et al. [46]	1 or N (fixed)	✓	✓	Deep learning (generative) + Algorithmic. Convolutional Network, point cloud fusion, silhouette fitting. Reconstruction losses, mask loss, adversarial loss and energy function.	Gray-scale image sketches from implicit views.	Point clouds + Meshes	✗	✗

Table 1. Summary of the state of the art in sketch reconstruction.

Table 2. Summary of the state of the art in realistic reconstruction.

Paper	Number of views	Automatic	Interactive	Technologies	Input	Shape	Materials	Lighting
Mildenhall et al. [48] (NeRF)	N	✔	✗	Deep learning (optimization). Multi-Layer Perceptron, positional encodings, hierarchical sampling, volume rendering. Image loss.	Images, viewpoints.	Implicit volume	Implicit	Implicit
Müller et al. [51]	N	✔	✗	Deep learning (optimization). NeRF extension, multi-level hash table encodings. Image loss.	Images, viewpoints.	Implicit volume	Implicit	Implicit
Wang et al. [77] (NeuS)	N	✔	✗	Deep learning (optimization). NeRF-inspired, Signed Distance Functions, modified volume rendering. Image loss and regularizations.	Images, viewpoints, optional masks.	Implicit SDF	Implicit	Implicit
Boss et al. [6] (NeRD)	N	✔	✗	Deep learning (optimization). NeRF-inspired, BRDF autoencoder, textured mesh extraction. Image loss and mask loss.	Images with potentially different illuminations, viewpoints, masks.	Implicit volume	Implicit BRDF	Spherical Gaussians
Zhang et al. [84] (NeRFactor)	N	✔	✗	Deep learning (optimization). NeRF initialization, refinement Multi-Layer Perceptrons, BRDF decoder, Physically Based Rendering. Image loss and regularizations.	Images, viewpoints.	Implicit surface	Implicit BRDF	HDR map
Zhang el al. [82] (PhySG)	N	✔	✗	Deep learning (optimization). Multi-Layer Perceptron, sphere tracing, rendering based on Spherical Gaussians. Image loss and regularizations.	Images, viewpoints, masks.	Implicit SDF	Spherical Gaussians + Implicit albedo	Spherical Gaussians
Yang et al. [80] (PS-NeRF)	N (with M lightings)	✔	✗	Deep learning (optimization). NeRF initialization, uncalibrated photometric stereo, refinement Multi-Layer Perceptrons, shadow-aware rendering. Image loss and regularizations.	Images, viewpoints, masks.	Implicit volume	Implicit Spherical Gaussians + Implicit albedo	Directional lights
Kim et al. [35]	N	✔	✗	Algorithmic (gradient-based optimization). Structure-from-Motion initialization, Lambertian rendering. Image losses and regularizations.	Images with potentially different illuminations.	Meshes	Vertices albedo	RGB Spherical Harmonics
Hasselgren et al. [27]	N	✔	✗	Algorithmic (gradient-based optimization). Deferred rendering. Image loss and Laplacian regularization.	Images, viewpoints, lighting, optional masks.	Meshes	PBR textures	✗
Gao et al. [17] (DefTet)	1 or N	✔	✗	Deep learning (generative) or Algorithmic (gradient-based optimization). Deformable tetrahedral grid, occupancy, differentiable rendering, encoder-decoder. Image loss (or reconstruction loss) and regularizations.	Images (or point clouds), viewpoints, optional masks.	Meshes	Vertices color	✗
Munkberg et al. [52] (NVDiffRec)	N	✔	✗	Deep learning (optimization). Deformable tetrahedral grid, Signed Distance Function, Multi-Layer Perceptron, deferred rendering. Image loss, mask loss and regularizations.	Images, masks, viewpoints.	Meshes	Diffuse textures + Specular textures + Normal textures	HDR map
Goel et al. [20] (SFT)	N	✔	✗	Algorithmic (gradient-based optimization). Structure-from-motion initialization, mesh colors, cyclic disjoint optimization, coarse-to-fine, remeshing, differentiable path tracing. Image loss.	Images, masks, materials segmentation, viewpoints, lighting.	Meshes	Roughness + Diffuse mesh colors + Specular mesh colors	✗

3. Specific technologies

In this section, we will discuss specific technologies relevant to our work. Firstly, we will present the principles of differentiable rendering, a key component of inverse rendering. We will cover the fundamentals of both local lighting approaches in the form of deferred shading and global lighting techniques in the form of path tracing. Both rendering methods, on their differentiable framework, will be necessary for our work.

Secondly, we will make a detailed presentation of NVDiffRec and its inner workings. This will give the reader a better understanding of the system as we will use it as a core component of our first proposal.

3.1. Differentiable rendering

Rendering techniques have been developed since the early days of computer graphics [12, 29, 31, 32, 39]. When talking about *rendering*, we refer to the process of transforming data structures into graphic visualizations, usually displayed on a screen. The most common application and the one relevant to us corresponds to transforming data structures representing 3D geometry and visual appearance properties into 2D images depicting such described three-dimensional scenes.

When considering the graphic representation of 3D scenes, we must refer to the foundation in this area: the rendering equation, seen in Equation 1. This equation characterizes the behavior of light in a scene, defining the light emitted on a surface point in a direction ω_o based on the integral on the hemisphere Ω around it of the incident light $L(\omega_i)$, the characteristic function of the surface's material $f(\omega_i, \omega_o)$, the surface's normal n, and the direction of the incident light ω_i. This equation must be solved to find the appropriate color for each scene point, conferring a realistic appearance to computer-generated three-dimensional scenes.

$$L\left(\omega_o\right) = \int_{\Omega} L\left(\omega_i\right) f\left(\omega_i, \omega_o\right) \left(\omega_i \cdot n\right) d\omega_i \tag{1}$$

However, estimating the equation's integral is highly complex, as it usually does not have a closed form. To approximate it, it is generally divided into diffuse and specular components, having been proposed many models to approximate both, such as Lambert, Cook-Torrance, and Disney, among others [8, 39, 86]. Moreover, both global and local methods have been developed. While local lighting approaches compute the color of a point based only on its local properties, global methods compute it given both the local properties and how the rest of the scene affects them. Consequently, global approaches account for complex phenomena such as object inter-reflection, transparency, or translucency.

While the process of rendering and the rendering equation have been widely studied and are well known, its inverse operation has received more attention in recent years. Often known as *inverse rendering*, this procedure aims to reverse the rendering operation: given the graphic 2D visualization generated, obtain information related to the underlying data structures that allow it. Inherently, inverse rendering represents a more complex task and often suffers from a higher uncertainty.

The recent development of inverse rendering techniques has been closely linked to the development and growth of auto-differentiation systems, such as PyTorch [92] or TensorFlow [93]. Thanks to

them, the most successful approaches have been found in developing differentiable rendering pipelines. By doing so, auto-differentiation allows reverting the rendering process and updating scene parameters from image metrics using gradient descent optimization techniques, as seen throughout Section 2.2.

Nonetheless, the task of defining a differentiable rendering pipeline is a challenging one. On the one hand, discontinuities may appear, making it difficult to differentiate the process without bias. On the other hand, rendering can be a resource-intensive and time-consuming process, making its differentiation very costly and tradeoffs necessary for using it on iterative optimization.

As seen in Section 2.2, there are many possible alternatives for differentiable rendering, depending on how the rendering equation is approximated. Some examples are volume rendering, used in models such as NeRF and NeRD, or spherical Gaussians, used in PhySG. However, these methods are tailored for the given task and representation, losing generality. In the following sections, we will detail the state-of-the-art main models for the generic differentiable rendering of explicit 3D representations.

3.1.1. Deferred shading

Deferred shading refers to the strategy in which all the spatially-varying attributes of the scene are stored in an image-space regular grid over which the shading function is later applied [40]. The base for the current lines of research for differentiable rendering using deferred shading is found in the work developed by Laine et al. [40].

They conceived differentiable rendering as a means for using modern machine learning with 3D geometry. Therefore, a pipeline capable of computing the loss gradient with respect to arbitrary scene parameters was desired. Furthermore, they aimed to leverage the well-known developments in real-time graphics to use existing pipelines. By doing so, not only desirable features such as programmable shading, parallelization, and correct outputs could be preserved, but also current hardware pipelines could be used.

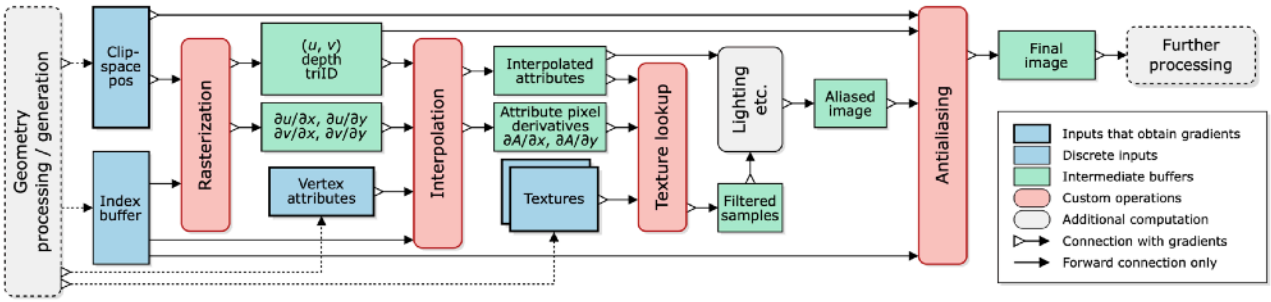

Figure 11. Differentiable rendering pipeline built using Laine et al.'s proposal [40].

To reach these goals, some design choices were put up front. The critical primitive operations that required differentiation were identified. By providing custom implementations for them, a modular design was accomplished. Geometry and textures were modeled as tensors for compatibility with existing auto-differentiation systems. Furthermore, deferred shading was leveraged to define shaders externally using efficient tensor operations. Finally, triangular meshes were considered to use the optimized rasterization in modern pipelines for graphics.

With these choices, four main primitive differentiable operations were defined, allowing for assembling differentiable rendering pipelines with them, as shown in Figure 11. The first was

rasterization, in charge of projecting the triangles to the image space. The forward of this module took as input the mesh's triangles and vertices in homogeneous clip-space coordinates of the form (x_c, y_c, z_c, w). Therefore, the user was in charge of obtaining these coordinates by multiplying the vertices coordinates $(x_v, y_v, z_v, 1)$ by the world, view, and perspective transformation matrices. From them, a 2D sample grid storing at each position the triangle ID, barycentric coordinates (u, v) in the triangle, and depth was obtained. Additionally, the derivates of (u, v) with respect to the screen coordinates (x, y) were also provided. By using OpenGL fragment shaders to obtain every output, the hardware graphics pipeline was used for the rasterization, ensuring accuracy.

Meanwhile, the backward of the rasterization received the gradient of the loss with respect to the barycentric coordinates and generated the gradient with respect to the vertices' coordinates, effectively updating the geometry. Similarly, the gradient concerning the derivates of (u, v) was also computed. A scatter-add operation was used to implement the backward to accumulate the gradients of the pixels on the correct vertices by using the triangle IDs.

The second primitive proposed was the interpolation. Given the grid of barycentric coordinates, this module's forward computed the grid cells' attributes by applying a weighted sum of the vertices' attributes through the barycentric coordinates. Moreover, as these attributes generally involve texture coordinates, this module also generated the Jacobian of all the attributes to be able to determine the texture filter footprint later. The backward of the interpolation worked similarly to the one of the rasterizer, using scatter-add to accumulate the gradients of each attribute into the barycentric coordinates.

The third defined module was the texture mapping module, which obtained texture values from the interpolated attributes. The implementation was similar to the interpolation module. First, a fractional mipmap level was selected, using the derivatives of the texture coordinate attributes to measure the major axis of the sample area. Then, trilinear interpolation from the four closest pixels of lower and upper resolution levels was applied. In this case, both gradients for the texture coordinate attributes and the screen-space derivatives of said coordinates were computed. Given the multi-scale nature of this module, the backward needed to revert the mipmap generation process, accumulating the gradients of all the levels into the finer one.

Finally, the last module introduced was the antialiasing system. The use of this module was crucial for differentiable rendering. While texture filtering allowed smoothness in the interior of surfaces, point sampling produced aliasing at silhouette discontinuities, making it impossible to compute visibility gradients. Antialiasing after the shading process converted these discontinuities into smooth changes, allowing for the estimation of gradients.

The forward step in the antialiasing module worked in two stages. Firstly, pixel pairs with visibility discontinuities were located by finding neighboring pixel pairs with different triangle IDs. Then, if the closest triangle to the camera contained a perpendicular edge to the pair crossing between their centers, said pair was considered to present a discontinuity. Secondly, for all pairs with discontinuity, blending was applied by considering the distances to the edge, as seen in Figure 12.

For the backward step, the discontinuity analysis performed in the forward was stored, avoiding the need for recomputing it. Then, for the aliased pixels, the gradient of the color was transferred to the vertex positions by scatter-add operations.

It is important to note that, although effective and efficient, this antialiasing method presents some limitations. Coverage is only estimated exactly with perfectly perpendicular edges, presenting coverage error for diagonal edges. Moreover, in the case of finely tessellated meshes, a higher error

could be potentially introduced as any arbitrarily shaped polyline could define the silhouette edge. Not only that but with finer enough levels, some triangles might be too small to get rasterized, causing some silhouette edges not to receive visibility gradient, slowing the optimization.

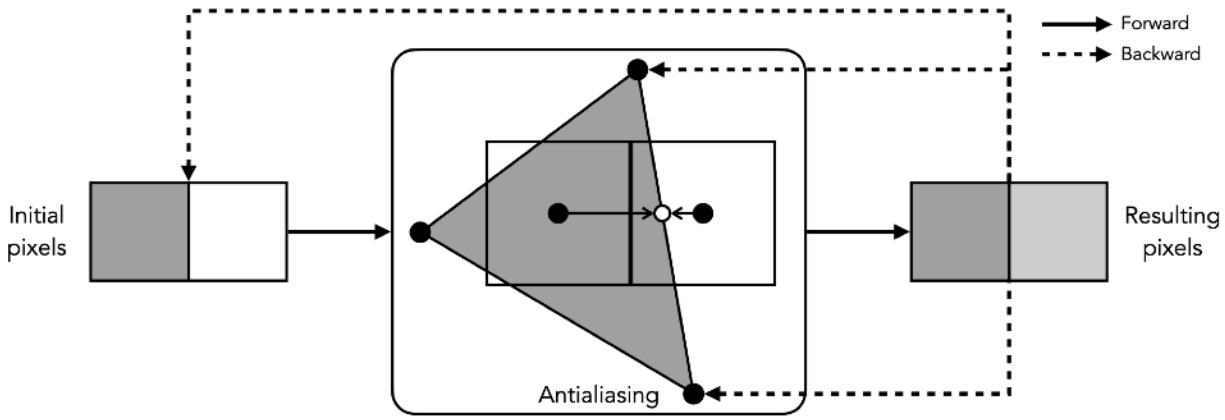

Figure 12. Scheme of the antialiasing operation by Laine et al. [40].

As we have seen, Lain et al. settled the bases for a differentiable rendering pipeline highly compatible with modern hardware for graphics and auto-differentiable systems. This allows for efficiency, quality, and flexibility, enabling the use of the four key modules presented to build full pipelines using auto-differentiable operations. It is important to note that, due to the deferred nature of the design, the shaders must work over the grids of attributes to provide a grid of pixels. This limits the rendering to a local lighting model in which the resulting colors can only depend on the local surface attributes. However, as a trade-off, the system's differentiable nature allows propagation of the loss gradient to arbitrary scene parameters, making this framework powerful.

3.1.2. Path tracing

Path tracing and ray tracing algorithms were proposed to provide a global illumination model capable of computing the color of a point in the scene based on global influences [32]. These algorithms approximate the rendering equation integral based on the principles of Monte Carlo sampling: the integral of any function can be computed as the average of *N* samples of the function multiplied by its range. When *N* tends to be infinite, the integral estimation tends to be exact.

Therefore, the ray tracing algorithm combines the principles of sampling and the behavior of light to compute the shading of the scene by shooting rays from the camera's pixels. As it can be seen in Figure 13, for each ray, the closest intersection with the scene is computed, using it as the source for newly recursively generated rays. Once the hierarchy of rays reaches its maximum depth or the end of the scene, the color of the first point is computed by combining all the subsequently sampled points along the hierarchy.

Following similar principles to ray tracing but aiming to reduce the variance, path tracing was proposed as a ray tracing algorithm in which the branching factor is reduced. Instead of tracing reflection and transmission rays at each intersection, only a single new ray is born from each intersection, as depicted in Figure 13. However, to keep the correct proportions between the different types of rays, the proper ray type to trace at each intersection has to be chosen based on the probabilities of the desired distribution.

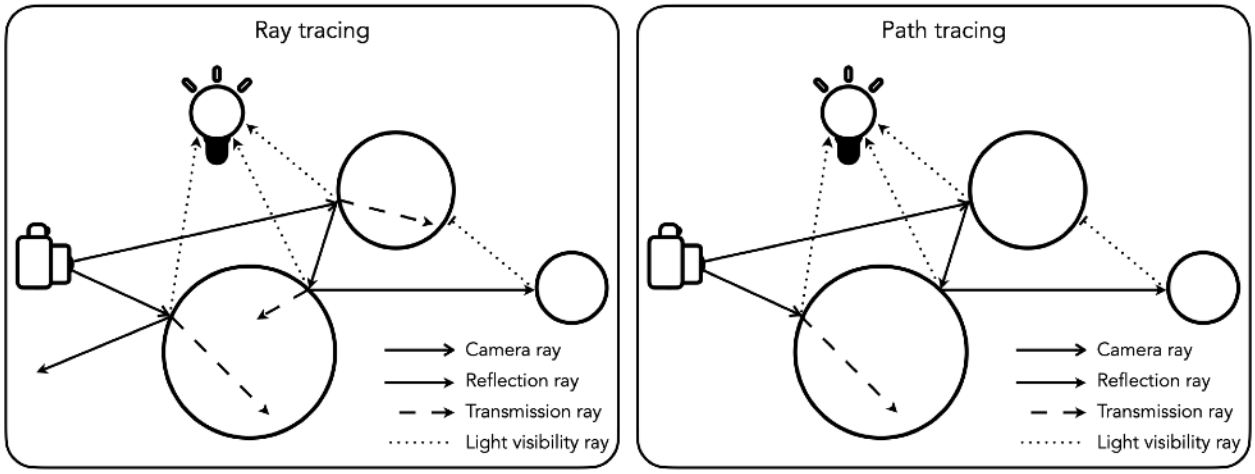

Figure 13. Schematic representations of ray tracing and path tracing.

Path tracing allows approximating the integration of the rendering equation through Monte Carlo sampling. However, when considering the differentiation of this process, the rendering integral presents discontinuities that make visibility parameters not differentiable at the object boundaries. These discontinuities in the screen space were already observed in Section 3.1.1, solved using antialiasing filtering. However, when considering the global illumination model, discontinuities also exist in the 3D space when computing the radiance, as objects may block the light received by the shading point. Li et al. [44] proposed a sampling technique for an unbiased differentiable path-tracing renderer to solve this issue.

Li et al. tackled the problem by locating the issue at the edges of the geometry. Even though Monte Carlo sampling could approximate the rendering equation integral and its derivative, the discontinuities made it impossible to capture changes caused by camera parameters or geometry translation. This was because, at the discontinuities, the derivatives were Dirac delta functions δ, depicted in Figure 14. Thus, traditional sampling techniques failed as they distribute the samples uniformly, being difficult to capture the changes in boundaries.

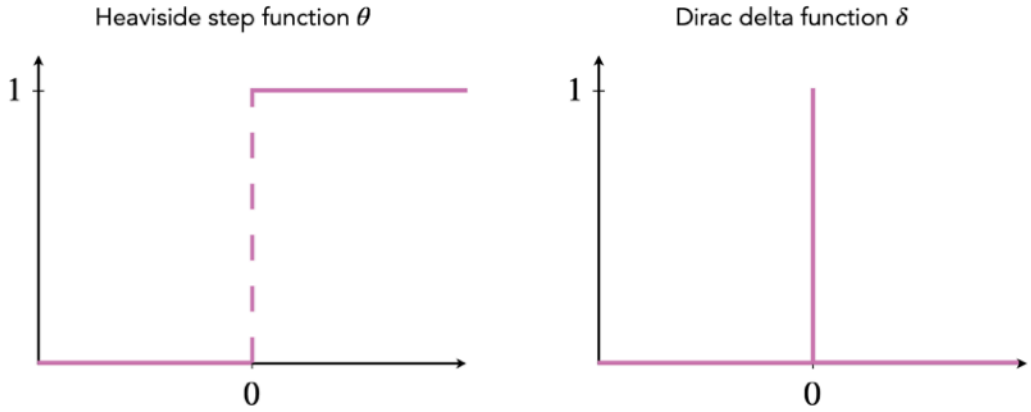

Figure 14. Heaviside step function and Dirac delta function.

Therefore, Li et al. proposed to model the edges causing discontinuities both in the screen space –primary visibility– and in the scene space –secondary visibility– as Heaviside step functions θ, shown in Figure 14. Therefore, using the edges α to control the step function, the space was divided into two half-spaces (f_u and f_l) separated by a discontinuity. Formally, the Heaviside step function created by a triangle edge in screen space can be expressed as in Equation 2, where x and y are 2D coordinates in the screen space.

This formulation allowed transforming the integration of the pixel colors as the summation of the integrals of the Heaviside functions regulating arbitrary functions f_i. Then, the objective of obtaining the gradient could be tackled by estimating the gradient of each Heaviside function.

$$\theta\left(\alpha\left(x,y\right)\right)f_u\left(x,y\right)+\theta\left(-\alpha\left(x,y\right)\right)f_l\left(x,y\right) \tag{2}$$

The analytic gradient of the step function gave the key result for the proposal. Using the product rule, the gradient of the integral of θ by f_i could be expressed as a sum of two integrations, as seen in Equation 3. On the one hand, the first part presented the integral of the derivative of the step function θ, characterized as a Dirac delta function δ. On the other hand, the second part contained the integral of the gradient of the arbitrary function f_i, whose content will be detailed later.

$$\begin{aligned}\nabla I &= \sum_i \nabla \int\int \theta\left(\alpha_i\left(x,y\right)\right)f_i\left(x,y\right)dxdy \\ &= \sum_i \int\int \delta\left(\alpha_i\left(x,y\right)\right)\nabla\alpha_i\left(x,y\right)f_i\left(x,y\right)dxdy \\ &+ \sum_i \int\int \nabla f_i\left(x,y\right)\theta\left(\alpha_i\left(x,y\right)\right)dxdy\end{aligned} \tag{3}$$

Equation 3 revealed that the gradient could be estimated through two Monte Carlo estimators. The continuous spaces corresponding to the second term could be estimated by the traditional pixel integral using auto-differentiation. Meanwhile, the first term represented the discontinuities and could be estimated by explicitly sampling at the edges. Therefore, an explicit sampling strategy was proposed to compute the boundary gradients, recording the difference between both sides of the edge E, following Equation 4. As in previous equations, f_u and f_l represent the half-spaces and α_i is the edge equation. Additionally, $\|E\|$ and $P(E)$ represent the length and probability of selecting the edge E, respectively. Both sampling strategies are depicted in Figure 15.

$$\frac{1}{N}\sum_{j=1}^{N}\frac{\|E\|\nabla\alpha_i(x_j,y_j)(f_u(x_j,y_j)-f_l(x_j,y_j))}{P(E)\|\nabla_{x_j,y_j}\alpha_i(x_j,y_j)\|} \tag{4}$$

Until this point, this formulation only covers the primary visibility. However, discontinuities can also appear in secondary visibility, caused by shading and shadows. Recalling Equation 3, Heaviside functions regulate arbitrary functions f_i. These functions can also contain additional step functions, representing the operations needed to compute the color of the image's pixels. In particular, they represent the integration of all the scene points m determining the shading of the evaluated point p.

When considering the shading integration, geometric silhouettes can block the influence of any point m over p, introducing additional discontinuities. Therefore, the approach followed on primary visibility was generalized to secondary visibility, allowing a similar factorization as Equation 3 for the shading integration in three dimensions. Consequently, given m respect to p, the shading gradient was computed by explicitly sampling the three-dimensional edges of potential blockers between them. However, sampling in secondary visibility is more involved than in screen space, as the shading point can be located anywhere.

To implement this specialized edge sampling, Li et al. proposed a scalable hierarchical sampling for an arbitrary viewpoint. Two volume hierarchies were generated: one containing all the edges belonging to a single triangle or to triangles non-smoothly shaded, and another containing the rest. Moreover, two traversals were applied. The first one aimed to detect the edges blocking the light as they had a more meaningful contribution, speeding up the process by excluding the non-intersecting volumes with the cone defined by the point and the light source. The second one sampled all the edges, computing their importance based on the length, distance, and response of the edge's material. Finally, importance sampling was also applied internally for each edge based on the material, light sources, and perspective distortion.

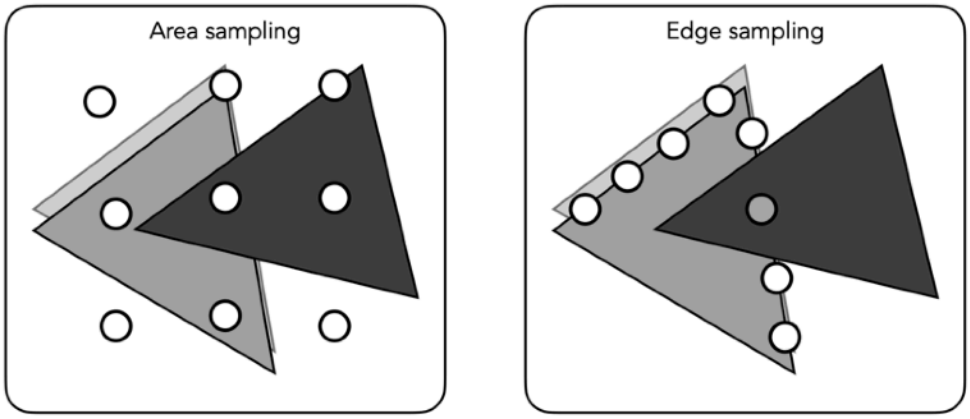

Figure 15. Sampling strategies proposed by Li et al. [44].

With this explicit Monte Carlo sampling of edges, the discontinuities could be explicitly captured, allowing for a differentiable pipeline concerning any arbitrary parameters in an unbiased way. Some restrictions were imposed, however, as triangle interpenetrations, point lights, and perfectly specular materials were not considered. Moreover, despite the implementation of importance sampling, the need for explicitly sampling the edges introduced a considerable bottleneck, increasing the differentiation's temporal costs.

Striking to offer a new model improving the efficiency of the differentiable path tracing in [44] while producing unbiased results, Bangaru et al. [2] proposed a formulation based on area sampling to avoid explicit edge sampling. They also based their proposal on a partitioned definition of the integration domain D, expressing the gradient over the image as the sum of the gradients of the integral for each subregion. In turn, similar to [44], they split the derivative of the integral for each subregion into a sum of two components: the interior derivative integral and the boundary derivative integral.

Until this point, the result reached by [2] presented similarities with [44], finding the solution in the estimation of the continuous regions plus the estimation of the discontinuities. However, Bangaru et al. reached this solution by applying the Reynolds transport theorem [63] to measure the change in the boundary. This allowed them to model the boundary integral as the rate at which the domain expanded or contracted over the edge, which enabled the application of the divergence theorem.

The divergence theorem relates the integral of a flux through a volume with the integral of the flux through the surface, making it possible to convert the boundary integral into an area integral using a warp field, avoiding explicit edge sampling as shown in Figure 16. Equation 5 shows the final formulation of the image gradient, where D' is the domain minus the boundary $D - \partial D$, $\vec{V}_\theta$ is the

warp field, θ represents arbitrary scene parameters, f is the rendering function, and ω is a 3D direction in the domain D.

$$\nabla I = \sum_i \nabla \int_{D_i(\theta)} f(\omega;\theta) d\omega = \sum_i \int_{D_i'(\theta)} \nabla f(\omega;\theta) dD_i'(\theta) + \sum_i \iint_{D_i'(\theta)} \nabla_\omega \cdot \left(f(\omega;\theta) \overrightarrow{\mathscr{V}}_\theta(\omega) \right) d\omega \quad (5)$$

However, the warp field had to be chosen appropriately to be continuous on D and closely match the true warp at the surface points. Therefore, Bangaru et al. proposed using the warp field obtained from the differentiated intersection function. Although this was consistent with the true values, it was not continuous. Therefore, a convolution over it was proposed using harmonic interpolation to generate inverse weights with the distance to the boundaries and make the warp continuous, as depicted in Figure 17. To avoid finding the closest boundary point, a simpler boundary test function tending to zero close to the boundary was used.

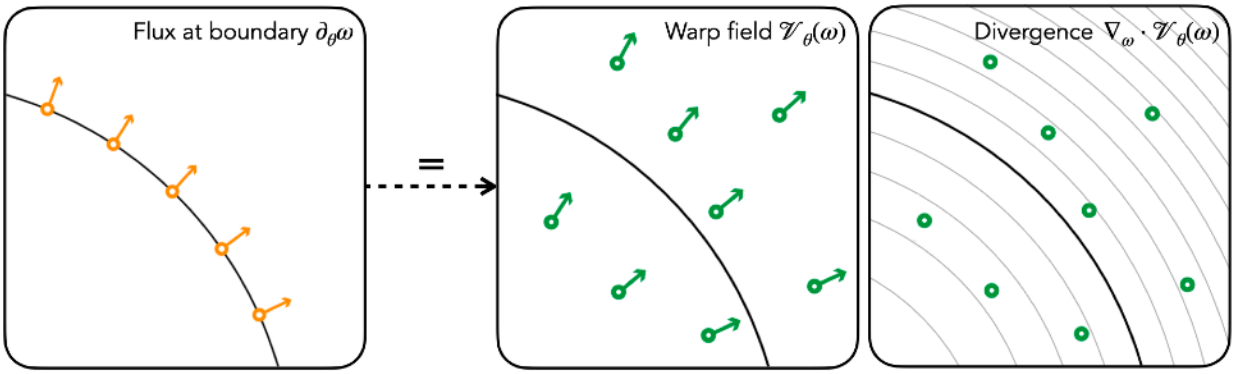

Figure 16. Equivalence between the boundary and area integrals [2].

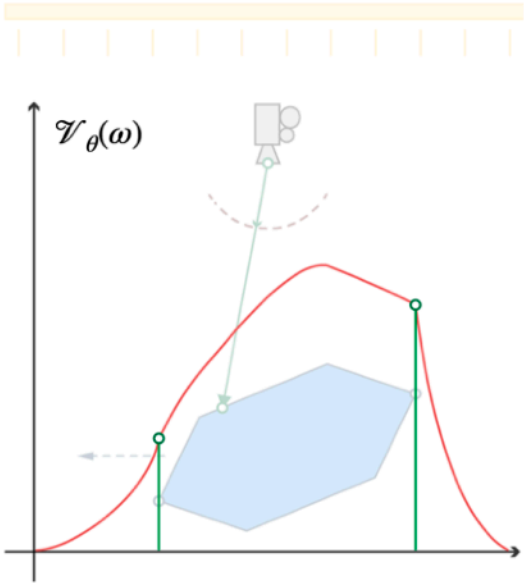

Figure 17. Graphic representation of the warp field proposed in [2].

Finally, with these ideas, a nested Monte Carlo estimator was established following the scheme in Figure 18. A secondary estimator was used for each sample generated through the primary estimator to compute the warp field. This Monte Carlo warp estimator fetched new samples, determining the boundary test and computing the convolution weights. Lastly, the warp estimation was obtained, making it possible to compute the gradient. To obtain an unbiased estimation, the warp field was determined with a high enough number of samples.

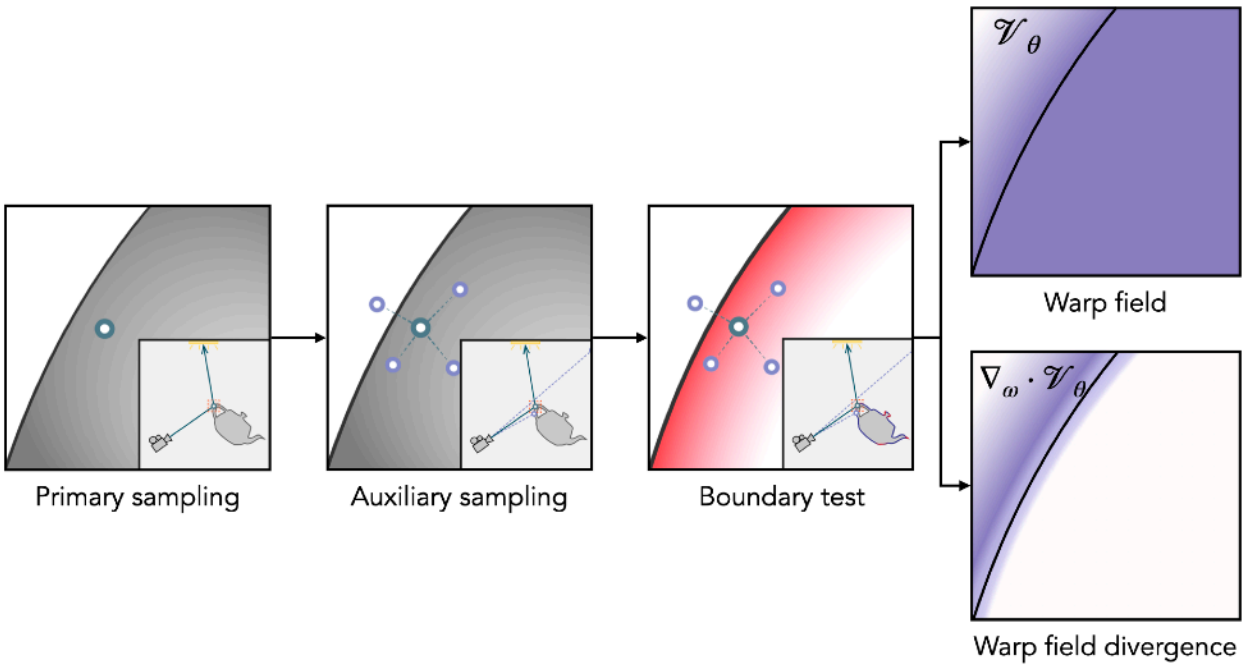

Figure 18. Nested sampling proposed by Bangaru et al. [2].

By converting the boundary integral into an area integral, Bangaru et al. provided a more efficient implementation unbiased inside some limitations, as the truncation in the number of samples used could lead to non-negligible bias depending on the memory constraints. Both the works of Li et al. [44] and Bangaru et al. [2] contributed to the development of the API *pyredner* [87]. Implemented in PyTorch, this library eases the use of differentiable rendering pipelines by providing the basic structures necessary for representing 3D scenes, such as meshes, materials, and cameras, as well as the required functions for rendering. Moreover, it is completely compatible with the auto-differentiation provided by PyTorch, allowing its integration and ease of use in machine learning systems with tensors.

3.2. NVDiffRec

As introduced in Section 2.2.2, NVDiffRec [52] constitutes a system capable of reconstructing textured 3D objects from multi-view realistic images. In this section, we detail its principles and architecture, settling the bases for our first proposal, where we will apply this system over non-realistic depictions.

3.2.1. Description

NVDiffRec's input comprises a set of images, masks isolating the target object, and the viewpoints associated with each image. The output is composed of a 3D triangular mesh, texture maps containing diffuse color, normals, and specular parameters, and an environment cube map. Therefore, shape, materials, and lighting are jointly recovered.

The objective of NVDiffRec was to generate reconstructions compatible with standard 3D content tools. This extended not only its utility and applicability but also relegated tasks such as simulating and relighting to specialized external systems. These principles are reflected in the design of the various components of the system, summarized in Figure 19.

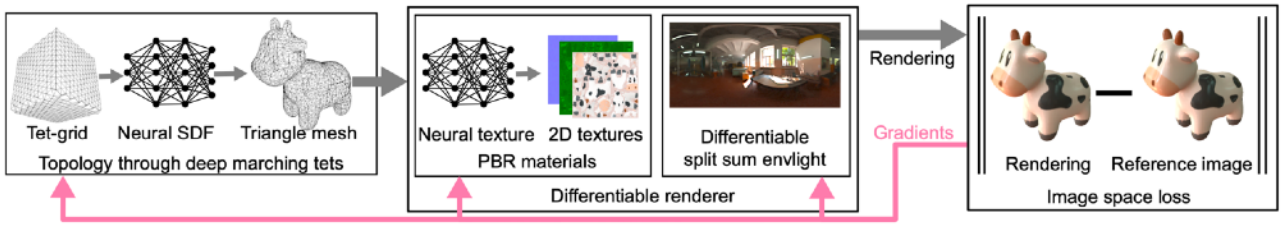

Figure 19. Summary of NVDiffRec [52].

First, the deformable tetrahedral grid from [17, 68] was used to represent the shape. Using Differentiable Marching Tetrahedra (MT) in each optimization step, the mesh was obtained and directly rendered. Therefore, explicit mesh optimization was performed. This compensated for the inherent discretization of the grid representation and its associated errors, allowing for a more closely refined shape.

Second, to texturize the meshes, volume textures were chosen. This choice was due to the joint optimization nature of shapes and textures. 2D textures require a mapping from the 3D vertices to the 2D space, potentially introducing discontinuities during the optimization when the number of vertices and their position is not constant. Meanwhile, volume textures allow accessing them through space coordinates. This provides a smooth variation with positional or topological changes.

The material model used for the textures was based on the PBR specification by Disney [8], continuing to look for compatibility. Therefore, materials with a diffuse term and a specular Trowbridge–Reitz (GGX) [73] lobe were considered. Three textures were used to characterize this material. Firstly, the diffuse texture contained the base colors. Secondly, the normal texture defined surface normals in tangent space. Finally, a specularity texture represented the roughness (green channel) and metalness (blue channel).

An MLP was defined based on [82] to represent the volume textures, encoding the diffuse, normal, and specular values based on the spatial position. However, image textures needed to be generated to export the results in a standard format. To this aim, the optimization progress was divided into two phases. While the first jointly optimized shape and MLP materials, the second aimed to refine the materials, fixing the topology and allowing only minor surface refinements. In this second phase, learnable 2D textures were automatically mapped to the mesh and initialized based on the values provided by the MLP, further refining them progressively.

2D feedback based on ground truth images and masks was used in the optimization. Key to this strategy was the use of differentiable rendering. For efficiency, deferred shading based on Lain et al.'s proposal was used [40], not considering reflections, refractions, or translucency. Moreover, the split sum approximation was used, dividing the rendering equation into two components: the integral of the Bidirectional Scattering Distribution Function (BSDF) under solid white lighting and the integral of the incoming radiance with the specular Normal Distribution Function (NDF). Both could be pre-integrated and stored, depending the first on the roughness and the cosine between the normal n and the incident light ray ω_i, while the second depended on the roughness and the direction of the outgoing ray ω_o. Equation 6 was obtained from the split, where Ω is the hemisphere around the desired point, L is the incident light in a direction, f is the characteristic function of the material, and D represents the surface's microfacet distribution function.

$$L(\omega_o) \approx \int_{\Omega} f(\omega_i, \omega_o)(\omega_i \cdot n)\, d\omega_i \int_{\Omega} L(\omega_i)\, D(\omega_i, \omega_o)(\omega_i \cdot n)\, d\omega_i \tag{6}$$

The use of this approximation defined the representation of the environment map as a differentiable cube map. While the base level described the pre-integrated lighting on the lowest roughness, the lower mip levels accounted for increasing roughness. These were computed through differentiable filtering, allowing to learn the base map directly. Additionally, a single low-res cube map was used for the diffuse lighting, sharing learnable parameters with the environment map. To enable differentiability, the mipmap generation had to be applied after each optimization update. Nonetheless, the split-sum approach allowed for speed-up computation, thanks to requiring only two texture lookups when compared to other methods.

Once rendering was applied over the mesh and materials given the view poses, feedback was generated to guide the optimization. NVDiffRec opted for an L1 loss for the images and a squared L2 loss for the masks. Additionally, multiple regularizations were applied. Firstly, light regularization was used to penalize the shift in environmental light color, as most of the real-world datasets contain neutral white light. Secondly, material regularization allowed for smooth material parameters. Thirdly, Laplacian regularization was only used on the second pass for maintaining the relative positions of the vertices and avoiding significant shifts. Finally, SDF regularization was only used in the first phase to avoid random structures in the models' interior, as they cannot be seen by the image or mask losses.

3.2.2. Architecture

The architecture used by NVDiffRec in its implementation is summarized in Figure 20. Throughout this section, we will provide a guided description of the different components detailed in this scheme.

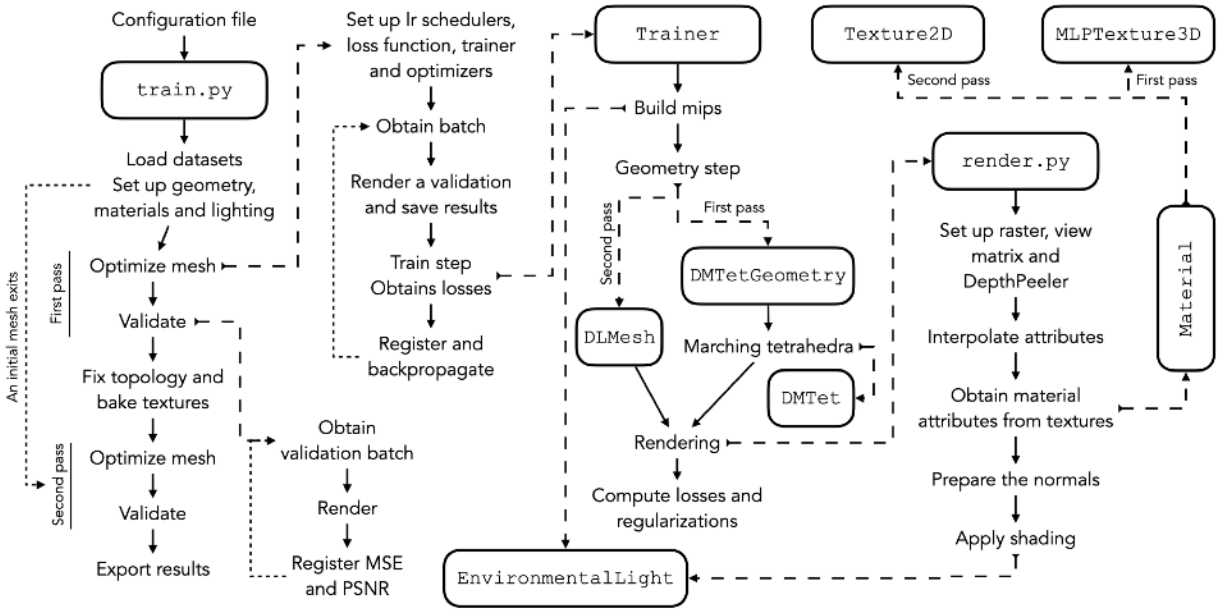

Figure 20. Architecture of NVDiffRec.

NVDiffRec was implemented using Python and PyTorch. The system is built on the *train.py* file, in charge of the required initializations, running the optimization loop, and saving the results. By running this program, a new reconstruction can be obtained. A wide variety of arguments can be provided, either directly on the invocation or through a configuration file. The main ones are the following, being noted with dashes the direct command arguments and, without them, the configuration properties that can be defined on a JSON file:

- *—config*: configuration JSON file.
- *-i*, *—iter*: number of iterations for the optimization of each one of the phases.
- *-b*, *—batch*: batch size used for optimizing.
- *-r*, *—train-res*: resolution of the images used for the optimization.
- *-tr*, *—texture-res*: resolution for the generated image textures.
- *-lr*, *—learning-rate*: learning rate used for the optimization. An individual learning rate for each phase can be specified.
- *-rm*, *—ref_mesh*: path to the input. It can be either a mesh file or a data folder.
- *-bm*, *—base-mesh*: path to a mesh file. If this argument is specified, this mesh will be used as the base for the optimization instead of the tetrahedral grid, skipping the first phase of the optimization.
- *envmap*: path to the HDR environment texture.
- *learn_light*: wherever the lighting should be optimized or not.
- *dmtet_grid*: resolution of the tetrahedral grid to be used. It can be set to 32, 64, or 128 by default.
- *mesh_scale*: scaling factor for the mesh.

Once the arguments are processed, *train.py* proceeds to execute the corresponding initializations. Two main initializations are performed. Firstly, the input data is loaded. This is carried out in multiple ways depending on the properties of *ref_mesh*. On the one hand, if this argument points to a 3D mesh file, this object is used to render reference samples from random points of view, using them for optimization.

On the other hand, if a folder is provided, the images and viewpoints are directly loaded. The loading process considers the folder's contents to keep compatibility with previous works, allowing training data from NeRF and NeRD. Independently of the input method, the images obtained integrate their masks on the alpha channel. At the same time, the viewpoints are specified through model-view and model-view-projection matrices for each image.

Secondly, the environment map is either loaded if no lighting optimization is performed or randomly generated otherwise. In either case, a cube map is obtained, represented as a tensor of 6 by 512 by 512 dimensions, and used to initialize an *EnvironmentalLight* object. Being a PyTorch module, this object presents a dual functionality.

On the one hand, it is responsible for generating the specular and diffuse mipmaps from the base learnable one. This is performed by average-pooling progressively to generate the specular mipmaps, locating the diffuse cube map at the lowest resolution. Once generated, the maps are pre-filtered with the GGX distribution using importance sampling.

On the other hand, it also implements the shading function. Given the global position p, normal $\vec{n}$, diffuse color d, roughness ro, metalness m, occlusion o, diffuse lighting l_d, specular lighting l_s, view vector $\vec{v}$, reflection vector $\vec{r}$, and precomputed BSDF integration, the shaded color c is obtained following Equation 7.

$$c = (1-o)\cdot\left(l_d(\vec{n})(d\cdot(1-m)) + l_s(ro,\vec{r})\left(BSDF_1(\vec{n}\cdot\vec{v},ro)((1-m)\cdot 0.04 + d\cdot m) + BSDF_2(\vec{n}\cdot\vec{v},ro)\right)\right) \quad (7)$$

Next, the reconstruction is executed. A new tetrahedral grid *DMTetGeometry* is constructed if no base mesh is provided, and a random MLP material is initialized. Then, after the first optimization loop, the estimated mesh is extracted, generating its texture coordinates through *xatlas* [89] and converting the MLP material to 2D textures. Finally, the second optimization loop is applied. If a base mesh was provided, the reconstruction jumps directly to the second phase with random 2D material textures. Once the optimization finishes, the reconstruction is exported as an OBJ file for the mesh, PNG textures for the material, and an HDR image for the cube map.

The optimization loop applied in all instances follows the same structure. First, the learning rate, learning rate scheduler following Equation 8, image loss function –being the default the log L1–, Adam optimizer, data loaders, and *Trainer* object are set up. Note that *warmup* in Equation 8 is a variable that takes a value of zero in the first phase and a value of 100 in the second phase. Then, for every training batch of the set repeatedly for the number of iterations, a training step is applied to the *Trainer*. From it, image losses and regularizations are obtained, being able to backpropagate the gradient to the geometry, material, and lighting parameters.

$$lr(\text{iteration}) = \begin{cases} \dfrac{\text{iteration}}{\text{warmup}} & \text{if iteration} < \text{warmup} \\ \max\left(0{,}10^{-0.0002(\text{iteration} - \text{warmup})}\right) & \text{otherwise} \end{cases} \tag{8}$$

The *Trainer* object is responsible for obtaining the losses to optimize the reconstruction. Internally, the construction of this object is simple, being a derivation of the PyTorch *nn.Module* class. Basically, it stores all the needed attributes for the optimization, setting up its parameters to all the learnable parameters in the scene. Then, for each forward pass, it generates the environment light mipmaps by calling the function in *EnvironmentalLight* and runs the function *tick* inside the geometry object to perform the rendering and compute the loss. Therefore, the *Trainer* object can be considered as a wrapper for these functions.

The *tick* function is critical to the process, as it calls the rendering function, allowing it to compute the losses and regularizations. Depending on the optimization stage, this function can be called in two different classes.

On the first pass, optimization is performed over the tetrahedral grid, represented by the class *DMTetGeometry*. This PyTorch module stores the vertices of the grid, the vertex indices of each tetrahedron, the SDF values of each vertex, and the displacement of each vertex, making the last two optimizable. This object presents a method for mesh generation, encapsulating the class *DMTet*, which implements the MT algorithm.

DMTet constitutes a functional class that transforms the tetrahedral grid representation into triangular meshes. The algorithm starts by identifying all the tetrahedrons located on the defined geometry's surface. In other words, the tetrahedrons whose number of vertices with positive SDF value is in the range]0,4[are selected. Then, all the unique edges are obtained, filtering out those whose vertex SDF values present the same sign. From the remaining, the position of the intersection of the edge with the mesh surface is computed as in Figure 8. Once the intersections are calculated, triangles joining them are defined using a look-up table containing all possible intersection cases. A total of 16 exist, being the different possible orientations and sign permutations of the non-empty cases in Figure 8. With the faces of the mesh defined, the algorithm performs a last step to generate texture coordinates for the vertices by evenly placing the triangles in a 2D space based on their ID.

Finally, *DMTetGeometry* also contains the classes required for rendering the geometry and the *tick* method. While the former runs *DMTet* to obtain the explicit mesh and calls the render function over it, the latter obtains the rendered images to compute the Mean Squared Error (MSE) of the alpha channels, the image loss over the remaining channels, and the regularizations. As this class corresponds to the first pass, SDF, albedo, visibility, and white balance regularizations are computed. The first penalizes the change of sign between the vertices of unique edges. The albedo and visibility regularizations penalize the difference between the values and the jittered values for smoother variation. Finally, the white balance regularization computes the average per-channel difference with the average intensity.

On the second pass, the optimization is performed over *DLMesh*. This class works directly over triangular meshes, making its vertex positions trainable. Its structure is very similar to *DMTetGeometry*, presenting mostly the same functions. In this case, the normals and tangents of the mesh are computed automatically before rendering. Moreover, as it is linked to the second pass, the used regularizations change, replacing the SDF regularization with a Laplace regularization for penalizing the vertices' change in relative neighboring positions.

Two main components remain to be detailed. First, the file *render.py* settles the differentiable renderer through hierarchically related functions. The function *render_mesh* corresponds to the head of the hierarchy, performing the full render and being the one used by *DLMesh* and *DMTetGeometry*. This function gathers the viewpoint matrices and converts the mesh vertices into clip space. Once in clip space, the scene is rasterized, interpolating vertex attributes such as the world positions, world normals, world tangents, texture coordinates, and texture coordinates derivatives, similarly to Section 3.1.1. Finally, from this raster, shading is performed using the shading function defined in the *EnvironmentalLight* map, generating the rendered image on a single pass. Composing operations can be performed afterward by using the alpha channel, as well as rendering other properties instead of shading, such as normals or specular parameters.

However, to apply shading, an important intermediate step is required. The material properties need to be obtained from the attributes in the raster. Depending on the pass, the materials, represented by the *Material* PyTorch module, can be accessed in two different ways. For the first past, the property *kd_ks_normal* can be sampled directly using the global position of the vertices, obtaining all the material properties at once. For the second pass, the texture coordinates are used to sample the individual diffuse, specular, and normal properties.

This behavior of *Material* is because it is designed like a dictionary, offering properties that the renderer can access. The key feature is that these properties represent textures as PyTorch module parameters that can be sampled using the appropriate coordinates.

During the first pass, the *kd_ks_normal* property stores an *MLPTexture3D* object. In turn, this object defines a hash grid positional encoding module in 16 levels connected to a sequential network of two hidden layers of width 32 with ReLu activation and a final layer of nine channels. Meanwhile, in the second pass, each material property is linked to a *Texture2D*. This PyTorch module represents a learnable image texture stored as a tensor and its associated mipmaps. For accessing the textures, both the texture coordinates and their derivatives are used.

Lastly, as shown in Figure 20, validation is applied after each training phase. If desired, this operation takes a validation set of images, masks, and viewpoints to generate renders of the current reconstruction and compare them with the ground truth. The MSE and the Peak Signal-to-Noise Ratio (PSNR) are used as metrics for the comparison, not contributing to the reconstruction while constituting a good reference to study its performance and progress.

4. Development

Once the relevant technologies and previous research have been presented, we proceed to detail our proposals. We aim to tackle the problem of 3D reconstruction from multi-view drawings. To this end, we make two proposals inspired by the use of inverse rendering techniques. Firstly, we will propose using NVDiffRec over the domain of non-realistic images. We will present a pipeline for using this existing system, highlighting the challenges for its use and how they can be approached. Secondly, we will leverage the architecture and principles proposed by Goel et al. [20] and introduce modifications for using it over the domain of sketches.

Therefore, this section will first define the problem we will be considering throughout this work and the justification for our approach. Then, we will detail both of our proposals. It is worth mentioning that both of them correspond with papers developed as part of our research. The first proposal corresponds with the work we presented at the 14th Asian-Pacific Workshop on Mixed and Augmented Reality [9]. Meanwhile, a second paper was elaborated from the development of our second proposal, having been sent to the 31st International Conference in Central Europe on Computer Graphics, Visualization, and Computer Vision. However, at the time of writing, the acceptance result of our second paper has yet to be published.

4.1. Problem statement and justification

We desire to obtain a system capable of automatically processing sketches to obtain 3D representations. In our work, we will only consider sketches represented as plain 2D RGB(A) images. Moreover, we will be working on the multi-view setup of this problem. This means that N sketches will be considered, depicting the sketched object from different points of view. To properly consider the multi-view framework, we assume the number of images to be $N \geq 3$. Additionally, in contrast to previous reconstruction efforts over sketches, we will also consider the color in them as a relevant property. This extends our potential scope, ranging from simple line sketches to more complete levels of colored drawn illustrations.

Given the multi-view references of a drawn target object, we desire to obtain a three-dimensional geometric representation resembling said object. However, drawings and sketches present inconsistencies due to the artist's skill and the subjectivity involved in the process, especially in the multi-view context. In most cases, this will make it impossible to find a single object capable of exactly matching and representing all the views. Therefore, rather than a perfect reconstruction of the target object, we will aim to obtain a reasonable approximation that contains the same visual essence and meaning as the desired object.

Similarly to [52], we will aim for compatibility with standard 3D content tools. Consequently, we will look for the generation of triangular texture meshes whose materials will be characterized through 2D RGB textures.

With this problem statement, we draw inspiration from the inverse rendering approaches developed for reconstruction over realistic images to build our proposals. The reasons behind this choice, striking a difference from the previous methods for automatic sketch reconstruction, are multiple.

Firstly, previous works on the same domain do not consider color, proposing techniques focused on obtaining untextured shapes. When considering the recovery of both geometry and materials, the closest references are found in the realistic domain.

Secondly, previous methods rely on deep generative modules to extract enough three-dimensional information to build the reconstructions. This implies that training is required to set up these modules, which can be troublesome as the amount of available paired data of hand-drawn sketches and 3D shapes is limited. This can lead to generalization issues as synthetic datasets are used for training. Moreover, these approaches tend to be trained for particular object classes, making the generalization to any class difficult. By leveraging the optimization techniques based on inverse rendering, we can tackle the problem through optimization, allowing for a generic system capable of working over differently styled sketches and potentially any object class.

Finally, the works in realistic reconstruction have shown that inverse rendering optimization is a powerful framework capable of delivering promising and good-quality results. However, as far as we know, this technique has never been applied to sketches directly. By approaching the problem through inverse rendering, we aim to determine if this technique is suitable for drawings, providing a new point of view for the domain of sketch 3D reconstruction.

Due to the nature of the techniques used in our proposal, some additional information will be needed to be able to perform the reconstruction. In particular, it will be required to augment the available sketches with information regarding the segmentation masks for the target object and the viewpoints corresponding to each drawing. We will propose generic alternatives for computing this information from images, aiming to apply them over sketches and drawings to see their viability. However, we will primarily focus on the reconstruction tasks, especially on our second proposal, leaving the deeper study of mask and view pose estimation from sketches for future work.

4.2. First proposal: using NVDiffRec

As we have presented in Sections 2.2.3 and 4.1, optimization-based approaches have a higher generality by nature when compared to deep learning methods. The limitations on their applicability for any domain reside in the assumptions taken during their design. Nonetheless, in general, as their operation is not limited by the number of cases seen in any training, they present a broader scope.

With these ideas in mind and given the promising results of NVDiffRec and their higher external compatibility, we aim to apply this system to non-realistic images. To do so, it was first necessary to determine the characteristics that the input images needed. As seen in Section 3.2, for being able to apply a reconstruction with NVDiffRec, it is required:

- A set of multi-view images of an object. In our case, the collection of plain 2D sketch images depicting a target from multiple points of view.
- A set of masks, one for each image, preserving the target object and hiding the rest. These will be provided as the alpha channel of the drawings in our case.
- A set of view matrices, one for each image, describing the position and orientation of the camera used to capture the image. In our case, the fictional viewpoint from which the sketch was drawn.

Given a collection of multi-view images without masks and viewpoints, such as is the case with illustrations, obtaining this information can be challenging. While the masks can be easily included in the design if we consider digital art, generating them for drawings already rendered or made traditionally is more complicated. Moreover, the need for camera information can be limiting. Given an illustration drawn from an arbitrary viewpoint by an artist, defining the exact mathematical point of view of the object is an even more challenging problem to overcome.

In this section, we present a workflow for using NVDiffRec with illustrations. As we consider a broad range of possible completeness levels and styles for the illustrations, we include two pathways in our workflow. On the one hand, we will present an automatic approach for mask and viewpoint generation using state-of-the-art techniques. On the other hand, we will also cover the manual methods. Figure 21 shows a summary of this workflow.

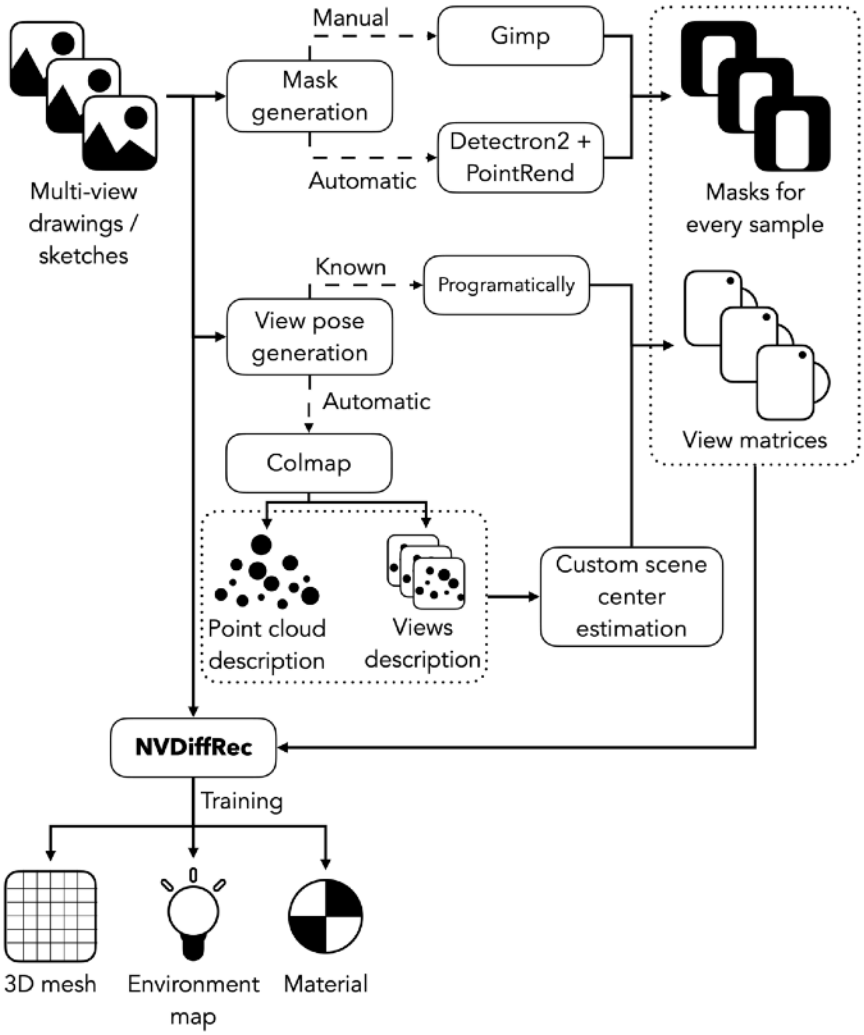

Figure 21. Summary of the proposed workflow.

Dashed arrows are conditional paths. Multi-view drawings and sketches represented as images are taken. First, mask generation is applied, either manually or automatically. Then view poses are generated either with prior information or automatically using COLMAP and adapting the results. Finally, images, masks, and views are used in NVDiffRec to obtain a textured model.

4.2.1. Generating masks

For identifying the target object that we want to reconstruct, it is required that we provide NVDiffRec with masks indicating which pixels belong to the target in each image. Given a set of 2D images that are not masked, we can follow two possible approaches to mask them:

- Process all images manually with an edition software such as Gimp or Photoshop to alpha mask everything except the target. This can allow more accurate results, with the tradeoff of being

much more costly from a user standpoint. For considerable amounts of data, it becomes unfeasible.

- Automatically analyze the images to identify the target object and mask it. This is also known as object segmentation. Image segmentation constitutes an open problem for which broad research exists with many good solutions, although none is perfect. This kind of automatic generation can be prone to error depending on the target object and can easily introduce faulty masks that can mislead the reconstruction. However, it allows the generation of masks for big volumes of images at a much lower cost.

The election of the method for generating masks will depend on the application, the complexity of the target object, and the number of images. When considering reconstructions from sketches or drawings, potentially created as concept art for designing characters or objects in the entertainment industry, the number of images available for a single reconstruction will be low. However, their complexity could be potentially high. In this case, manual mask generation may be effective, especially for artists already working with digital drawing tools, requiring less effort. Despite it, when a high number of reconstructions need to be generated and, therefore, many objects are illustrated, it may still be convenient to rely on automatic mask generation.

When considering automatic mask generation for our experiments, we opted for using Detectron2's API [78] and the PointRend model [36] to identify recognizable objects and their segmentation masks. By joining all the segmentations, we generated the mask of the image. This approach has the inconvenience of occasionally introducing outlier objects in the masks or masking out the target. Therefore, we removed from the set those images that, after masking, were empty.

4.2.2. Generating view information

NVDiffRec uses rendering to generate images comparable with the given samples to obtain feedback and guide the optimization. Therefore, knowing the view matrix associated with each sample is necessary to render it correctly from the same viewpoint relative to the object.

Given that we have a set of multi-view images with no camera information, we need to generate the view matrices in a way that is consistent with the target object, keeping the transformations between views compatible with the images. In this case, we can also identify two different approaches:

- If the images follow a known uniform transformation relationship between them, we can programmatically simulate this transformation for each image and generate the corresponding view matrices.
- When the images do not follow a known uniform distribution, estimating the view matrices can be a challenging problem. Indeed, this falls under the umbrella of research areas such as Structure-from-Motion (SfM) [67] and camera pose regression [66]. Therefore, to automatically generate image pose information, we must face an open problem and recur to the developed tools in these areas.

Again, choosing the approach to follow depends on the use case we face. Throughout Section 5.1, we will present three use cases that will exemplify the application of the different strategies we have described. However, before that, we consider it appropriate to give a more detailed insight into how camera views can be obtained.

When considering the first case, a clear example we will be facing in Section 5.1 is when the provided sketches depict the turn-around of an object. Turn-around animations are a common way of clearly presenting objects. By displaying the target through multiple views, turning the camera around the vertical axis of the object and tracing a circle of a given radius R around it, we can show all its geometric and color features. Especially when looking at character design, it is common to find turn-around depictions of such characters using a few views around them. This special case of multi-view setup allows for an easier camera pose estimation. Assuming that the angle turned around the vertical axis is constant between images, we can compute the view matrix by progressively turning the camera around the scene's center at a fixed radius and incrementally for each image. In this way, the turning angle in each step will be computed as 2π radians divided by the number of images.

In contrast, when the source images available do not follow a known uniform transformation, computing the camera for each image following the previous approach seems unfeasible. Therefore, the second approach needs to be used. For our experiments, we decided to use COLMAP to automatically generate viewpoints, given its widely available documentation, good performance, and ease of use.

As seen in Section 2.2.4, this system allows processing large amounts of images and using their key points to find the spatial relations between them, generating a point cloud representation of the scene. As a result, from multi-view images, COLMAP estimates the camera pose of each image. However, the compatibility between COLMAP and NVDiffRec is not direct.

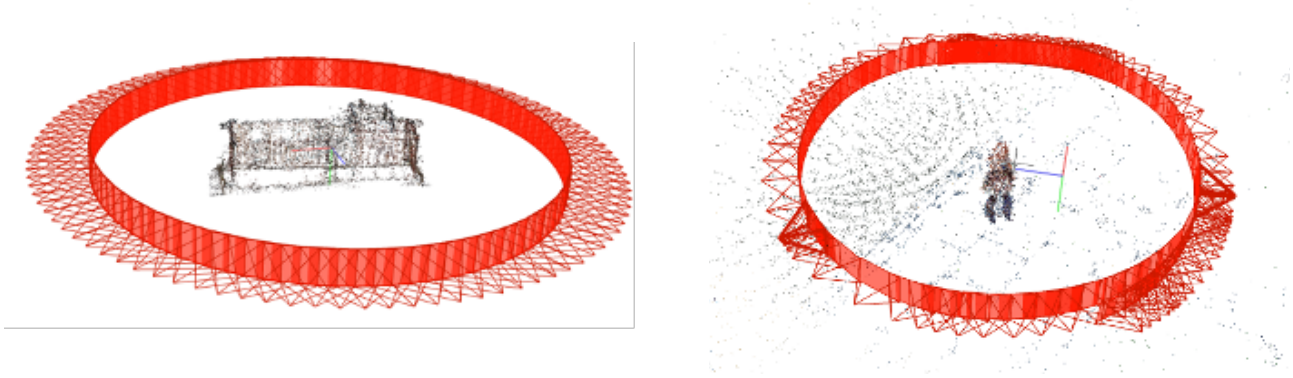

Figure 22. COLMAP center estimations for different view distributions.

Left, COLMAP estimation of a uniformly distributed scene view. The origin falls at the center of the object. Right, estimation of a non-uniformly distributed view in a different scene. The origin does not fall in the object.

Firstly, COLMAP and NVDiffRec have different coordinate systems, being the Z and Y axis inversed in one respect to the other. Secondly, COLMAP also estimates the origin of the coordinates of the scene. Given that this point depends on the camera distribution, as shown in Figure 22, the center generally does not match the target's center unless a uniform view distribution is given. This causes a disparity between the render and the ground truth because NVDiffRec places the mesh at the origin, but in the estimated view by COLMAP, the target is not at the origin. Therefore, to properly use the views estimated by COLMAP in NVDiffRec, we must determine the object's center in the coordinate system estimated and use it as a new origin. We explored two solutions to accomplish this.

On the one hand, if we do not know the nature of the object and its location in the different views, we can only consider the view poses to estimate the real origin. To do so, we can assume that, as the

samples capture a single object from different viewpoints, the camera positions are approximately distributed on the surface of a sphere of radius R around the target.

Given these considerations, we can locate the new origin by finding the sphere that most closely explains the camera positions. Moreover, we can also guide our decision by considering the cameras' looking directions. To solve this problem, we designed a Greedy Randomized Adaptive Search Procedure (GRASP) algorithm that, given camera positions and looking directions, tries to approximate the desired sphere by heuristically generating solutions and saving the best one. Section 4.2.3 details this algorithm.

On the other hand, we can use additional information to get a better estimation. In some cases, such as the one we will present in Section 5.1, we know that the target will always be located in a concrete region of the screen. Therefore, we can assume a known bounding box inside the images that always contains it. This is a reasonable assumption as, when taking multi-view samples of an object, it is usually kept in the same area of the image. Moreover, a single bounding box could be easily defined by a user.

With this bounding box (BB), we can use the information generated by COLMAP to filter the point cloud of the scene and then compute the center of this filtered version. Filtering is archived by applying a voting scheme such that each key point inside the BB in an image receives one vote. After analyzing all the samples, we can preserve the K most voted key points. Therefore, the center can be computed as the weighted average, using the votes as weights.

This approach is intuitive as key points of the target should be more commonly seen. However, it can be limited by the requirement of specifying a bounding box depending on the use case.

Finally, we can obtain a new averaged center using both estimation methods. For this alternative, we proposed applying a weighted average between the centers of each estimation, computing the required weights using Equation 9. In this equation, given C as the set of all cameras with look-at vector $\vec{v}$ and position $\dot{p}$, the cosine of the angle between the view and the direction to the point $\dot{x}$ is measured for each camera. Then, this measure is inverted and accumulated, decreasing the weight with the increase of accumulated value. Therefore, we compute the weights by giving higher importance to the points better aligned with the views and presenting a smaller angle with the view directions.

$$w(\dot{x}) = \left(\sum_{(\vec{v},\dot{p}) \in C} 1 - \left(\vec{v} \cdot \frac{(\dot{x} - \dot{p})}{\|\dot{x} - \dot{p}\|} \right) \right)^{-1} \tag{9}$$

4.2.3. GRASP algorithm for sphere estimation

Finding the sphere that best describes the view distribution of a set of cameras constitutes an optimization problem for which exact methods would be unfeasible in big datasets. Therefore, we try to find an approximation in a reasonable time using a Greedy Randomized Adaptive Search Procedure (GRASP) algorithm.

Our implementation reduces the sphere estimation problem to the task of finding four cameras whose positions describe a sphere that approximates the distribution of all the views. Consequently, our GRASP can focus on generating solutions formed by a sequence of four camera positions. After

obtaining these points, the center and radius of the sphere can be obtained by applying the general equation of the sphere.

Algorithm 1 presents our GRASP proposal for sphere estimation. Following the general scheme of this type of algorithm, every iteration has two phases:

- A constructive phase in which N solutions are generated. Each solution is built step by step, adding progressively new elements (camera positions). The first element is picked randomly among all the points. Then, every subsequent element is added semi-randomly, considering the cost of every remaining option as the inverse of the sum of the distances to each point in the current solution. In this way, we favor a more dispersed set of points. The best solution of the N generations is stored if it improves the current best solution.

Algorithm 1. GRASP Sphere Estimation.

```
1:  function sphereEstimation(points, dirs)
2:      distances ← distanceMatrix(points)
3:      max_r ← 2 · max(distances)
4:      best_sol ← ∅, best_cost ← ∞
5:      for _ ← 1 to max_iterations:
6:          for _ ← 1 to N:
7:              sol ← { random(points) }
8:              for _  ← 1 to 3:
9:                  cands ← points ∉ sol
10:                 costs ← dists(cands, points)^-1
11:                 cmin ← min(costs)
12:                 cmax ← max(costs)
13:                 cands ← { c ∈ cands | costs[c] ≤ cmin + α · (cmax – cmin) }
14:                 sol ← sol ∪ { random(cands) }
15:                 if cost(sol) < best_cost and radius(sol) < max_r:
16:                     best_sol ← sol
17:                     best_cost ← cost(sol)
18:         for _ ← 1 to max_depth:
19:             neighs ← getNeighbors(best_sol)
20:             for sol in neighs:
21:                 if cost(sol) < best_cost and radius(sol) < max_r:
22:                     best_sol ← sol
23:                     best_cost ← cost(sol)
24:                 else:
25:                     break
26:     return sphere(best_sol)
```

- A local search phase in which the algorithm tries to improve the current best solution by exploring its neighborhood. For generating the neighborhood, we take the indices of each point in the current solution and displace them randomly and circularly, one value up, down, or maintaining the value. With the new indices, we can find a neighboring set of points. In all iterations, M local solutions are generated. If none is better than the current solution, the search stops. Else, the best replaces the current, and the exploration continues up to the maximum depth.

Once the algorithm reaches the maximum number of iterations, the center and radius of the sphere described by the best solution can be obtained. Note that we define the best solution as the one that allows obtaining a sphere that minimizes Equation 10, where $\dot{x}$ is the sphere's center, r is the radius, C is the set of all cameras described by a look at vector $\vec{v}$ and a position $\dot{p}$, and w is defined in Equation 9. As it can be seen, this function measures how well-aligned the center of the current solution sphere is with the viewing directions and how consistently placed at a distance r it is. It is important to point out that, to avoid the sphere growing excessively, the radius of any solution is limited for it to be considered a valid solution. In our experiments, we used a fixed number of iterations of 1000, N of 20, M of 60, max depth of 50, α of 0.6, and the maximum allowed radius to double the maximum distance between cameras.

$$c(\dot{x}, r) = 0.4 \sum_{(\vec{v},\dot{p})\in C} |\|\dot{x} - \dot{p}\| - r| + w(\dot{x})^{-1} \tag{10}$$

4.2.4. Implementation

For implementing the automatic paths of the workflow proposed in Figure 21, we extended the publicly available implementation of NVDiffRec with two additional dataset managers inheriting from the NVDiffRec class *Dataset*. Firstly, we integrated a newly *DatasetSketchTurnAround* class for loading the datasets involving the turn-around multi-view case. Secondly, a class *DatasetColmap* was added, integrating the automatic mask generation and view pose estimation of the images.

The class *DatasetSketchTurnAround* presents a basic behavior similar to the rest of the dataset managers in NVDiffRec. After loading N image sketches as reference samples, the corresponding model-view matrix for each image is generated. This is possible by a fixed translation of the camera of two units in the Z axis towards the viewer, and a rotation of the said camera around the vertical axis (Y), following an angle of $2\pi/N$ multiplied by the ID of the image in the sequence, being the first one the ID zero. Finally, an orthographic camera is used to generate the model-view-projection matrix due to the common nature of sketches. As far as we know, NVDiffRec has not been tested under orthographic views before our work.

The class *DatasetColmap* involves a longer process in its preprocessing step. We extended the program's flags to apply additional operations after loading the images in this class:

- *—use-bb*: this corresponds to the second alternative to the scene center estimation. When this option is true, the flag *bounding_box* is read as it is supposed to store the bounding box inside the images. The bounding boxes are defined through the upper-left and lower-right corner pixel positions.
- *—center_estimation*: this argument allows the choice of the center estimation technique. The GRASP Sphere Estimation algorithm is used if "grasp" is provided as a value or a BB was not

provided through —*use-bb*. If "averaged" is used, then the average center between the GRASP and the BB estimation is utilized. Finally, the BB estimation is used otherwise.

Once loaded the images, *DatasetColmap* processes them to generate automatic masks by using the *detectron2* API [78] and the pertained PointRend model provided by it, as described in Section 4.2.1. Optionally, this estimation can be improved using the bounding box by directly masking out all the content outside it, as well as by deleting the samples completely masked inside the bounding box region after the segmentation.

From all the images without masks, COLMAP is executed over them using the *pycolmap* API [88] to extract features, match them, and generate the view poses and the scene point cloud from them. After properly transforming the points and views to NVDiffRec's coordinate system, the GRASP or the BB center estimations are applied and optionally averaged. While the first works only over the estimated cameras, the second uses the point cloud and its corresponding protections over each image provided by COLMAP. Once the new center is found, all the camera views are remapped to the new coordinate system origin, being able to use them for the reconstruction.

4.3. Second proposal: modifying the SFT architecture

Our first proposal aims to study whether the state-of-the-art reconstruction techniques over realistic multi-view images can be applied directly over illustrations. As seen in Section 4.2, we propose using NVDiffRec due to its promising reported results, compatibility, and generality of use.

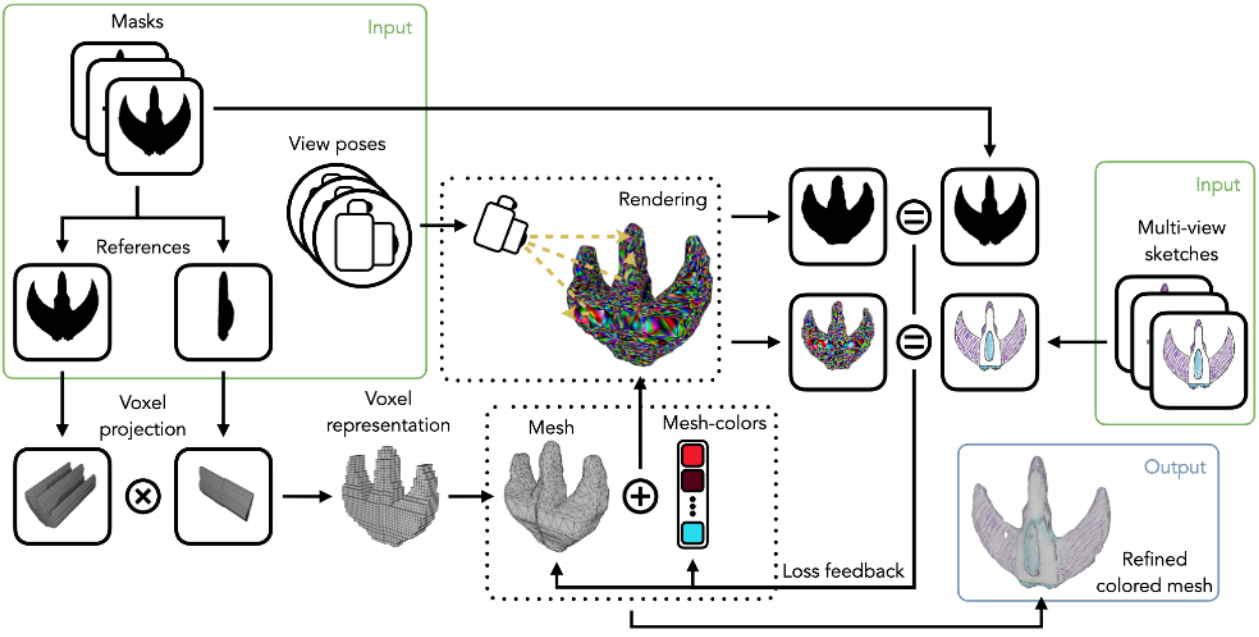

Figure 23. Summary of our second proposal.

Given multi-view sketches, masks, and view poses, an initial mesh is estimated using projections into a voxelated space. The initial mesh and random colors are refined through differentiable path tracing until obtaining the reconstruction.

However, even though the nature of this kind of optimization-based approach allows for applying it over any kind of image by providing the required inputs, it is important to keep in mind that we are working outside the intended domain. Even though the system may allow enough generality to be used outside its original target domain, its core components will always be better tailored and tuned

to work on it. This implies that, over other domains, the expected performance will be generally lower than the original one, being equal in the best possible scenario.

Moreover, some features may not be required for the new domain. In the particular case of drawings and sketches, NVDiffRec includes components such as the reconstruction of specular and environment maps, which are not really required in our case. Illustrations, especially when made as design references, generally present a constant illumination and do not realistically reproduce specularities. Including the estimation of these parameters in the optimization raises the level of uncertainty and the number of variables, increasing the difficulty of the task and wasting resources.

With these ideas in mind, it may be convenient to use the principles of inverse rendering optimization to build a more tailored solution for the domain we are tackling. As seen throughout Chapter 2, this scheme has not been previously used directly over drawings as far as we know. Therefore, we propose developing an inverse rendering optimization system for 3D reconstruction over multi-view illustrations by modifying the proposal of Goel et al. [20]. Even though we will focus mostly on sketches and flat-closed drawings in this proposal, it is important to note that the introduced system could also be used with higher-level illustrations.

As mentioned in Section 2.2.2, Goel et al. proposed a system for optimizing meshes from multi-view realistic images, which we will refer to as SFT. In this section, we detail the modifications we propose to apply over SFT to make it suitable for the domain of sketches and colored drawings, also gathering inspiration from some aspects of NVDiffRec. Our modifications fall into four areas:

- Sketches do not generally require realistic materials or complex lighting, as we will further see in Section 5.1. Therefore, we replace the BRDF materials with a single purely diffuse material and fix the environment map to completely white. Consequently, the system's input is reduced to multi-view plain image drawings, masks isolating the target –assumed to be represented by the alpha channel of the provided images–, and the view poses of each image, similarly to NVDiffRec. Even though requiring the view poses keeps being a limiting factor when working with sketches, the need for camera information is inherent to inverse rendering approaches. To be able to focus on the reconstruction, in this instance, we will assume them as known, being already provided to the system.
- Simultaneous optimization of mesh and materials is possible, as shown by [27, 35, 52]. We replace the alternating scheme proposed by Goel et al. with joint optimization, followed by a long-tail refinement of the colors inspired by the double-phase setup of NVDiffRec. Even though disjoint optimization can allow finer geometric detail [20], the detail requirement in sketch reconstruction is generally lower, given that inconsistencies difficult the correct capture of finer details. Therefore, simultaneous optimization can lead to good results with fewer steps.
- Goel et al. reset the material to a neutral grey after remeshing, discarding the estimated color in the previous material phases. This was done to avoid the propagation of any possible error during the optimization. However, we argue that completely discarding the color reduces the system's efficiency and increases the required optimization steps. In contrast, we propose a resampling scheme to recover the color partially, allowing the following steps to build upon the previous base color while still reducing the impact of possible previous errors.
- While Goel et al. used a single Mean Squared Error loss over the images to optimize either geometry or color, we guide the refinement process using split losses for shape and color, as well as regularizations. The use of split losses is motivated by our joint estimation of shape and color. By splitting the loss, we can tailor the function guiding each component, reducing the impact of

inconsistencies and joint uncertainty. Moreover, the use of regularization is motivated by the higher inconsistency of our task, requiring further guidance toward reasonable solutions.

Nonetheless, our system also presents common elements with Goel et al.'s work. These are the refinement over an initial mesh, using the mesh colors data structure proposed in [81], and using remeshing for solving potential artifacts and degenerations. The following sections present the solution in more detail, and Figure 23 summarizes it.

4.3.1. Optimization scheme

Given the sketches, masks, and viewpoints, an initial mesh is optimized to represent the sketched object. Goel et al. experimented with several mesh initialization techniques, such as voxel carving or using COLMAP to generate an initial mesh from the estimated point cloud structure, reporting the best performance with the latter. However, sketches and flat-colored drawings usually contain few key points, rendering COLMAP generally inadequate to obtain meshes from them, as further discussed in Section 5.1 and Chapter 6. Instead, we use a simple visual hull estimation based on parallel projections from a given subset of sketches into a voxelated occupancy space.

Given a subset of reference views the user provides, we project their footprint orthogonally into a voxelated space following the viewing direction. Therefore, the total shape can be computed as the intersection of all the voxelated projections. Then, a mesh is obtained through marching cubes, remeshing, and simplification. It is important to note that, to keep the resolution of the mesh low, the images are scaled down before projecting their silhouette.

To encode the colors linked to the mesh, a mesh colors data structure is used instead of textures [81]. With this representation, colored samples can be associated with a triangular mesh by storing them in a single vector. The number of samples per triangle depends on the mesh colors resolution R, which defines the level of simulated subdivision inside the triangle. The lowest resolution is $R = 1$, used when there is only one sample at each vertex. Augmenting R behaves like imaginarily tessellating the triangles adding more color samples, increasing the colors per edge to $R - 1$ and the colors per face to $(R - 1)(R - 2)/2$. We fixed R to three in our system.

Therefore, given the triangle ID t, the resolution R, and the barycentric coordinates (i, j) of a sample with $0 \leq i \leq R$ and $0 \leq j \leq R - i$, the index c of the sample in the color vector is computed with Equation 11.

$$c = \frac{(R+1)(R+2)}{2}t + \frac{2R - i + 3}{2}i + j \tag{11}$$

As we can see, this representation allows us to directly link the triangles to their colors, making it possible to perform sampling with barycentric coordinates. By using this representation, we can directly optimize the color vector as the mapping is coherent and direct to the mesh, independently of the geometry. Therefore, like in NVDiffRec and SFT, we avoid the need to optimize both colors and mappings between mesh vertices and image textures.

With this setting, the refinement involves optimizing the mesh vertex positions and the color vector. These parameters compose a 3D scene containing a single object in the origin that, when rendered from the viewpoints provided, is converted into 2D images. These images can be, in turn, compared with the references using loss functions. Finally, the differentiable rendering allows using gradient descent optimization to update the parameters jointly.

4.3.2. Losses

Losses are essential for our system, as they will be in charge of guiding the optimization toward a desirable reconstruction. Therefore, they must account for the properties of the task and the desirable features of the result. Our objective is to obtain a colored 3D triangular mesh that resembles the sketched object. However, sketches are not consistent descriptions but interpretations of the world. Consequently, we do not aim to find a replica but a reasonable approximation. In this way, the losses should be designed to allow enough flexibility to tolerate inconsistencies inside the range of reasonability while still capturing the features of the target object.

The main losses must inform the characteristic shape and colors of the sketched target. Instead of capturing both with one loss [20, 35], we establish dedicated losses. This not only allows better tailoring but also helps to reduce uncertainty in joint optimization as the changes in color and shape are explicitly separated, easing the process. Our proposed losses are:

- Color loss. Color details in sketches are inconsistent when considering the multi-view case. Sketch lines have two uses: conveying the surface color detail and representing geometric features. Both produce color feedback when applying image metrics, but only the former corresponds to true color information. Moreover, the second type is inconsistent between views. This can be visualized when considering outline lines, as they change with the silhouette of the target throughout different views, always tangent to the camera. We use a Laplacian pyramid loss [4] for comparison at different resolution levels to deal with these issues. Coarser levels inform the general color. Meanwhile, finer levels reinforce consistent lines, while inconsistent lines are overtaken by coarse color feedback. In this way, we can capture the general surface colors while ignoring the geometry-related lines. Equation 12 presents the formulation for our color loss where I_P and I_T are the rendered and reference images; K is the number of levels (which we fixed to three); G is the Gaussian filter function; $|\,I\,|$ and $D(I)$ are the number of pixels and channels of the image; and I^l represents the image scaled down by a factor of l^{-1}.

$$L_C(I_P, I_T) = \sum_{i=1}^{|I_P|D(I_P)} \frac{\left|I_{P_i}^{K+1} - I_{T_i}^{K+1}\right|}{|I_P|D(I_P)} + \sum_{l=1}^{K} \sum_{i=1}^{|I_P|D(I_P)} \frac{\left|I_{P_i}^{l} - G\left(I_{P_i}^{l}\right) - I_{T_i}^{l} + G\left(I_{T_i}^{l}\right)\right|}{|I_P|D(I_P)} \tag{12}$$

- Silhouette loss. Due to the inconsistency in color and general lack of shading in sketches, the silhouette defined by the masks is the primary source of shape information. We can capture this information using the Mean Squared Error between the reference and rendered masks. Even though the outline can also present inconsistencies, this loss balances the feedback among the references, averaging them. This is our intended behavior as, when considering multi-view sketches, especially when dealing with character design, it is possible that some parts of the target present a slightly different pose. Choosing which one should be the right one is highly difficult and ambiguous, as any of them would correspond with the desired object. Therefore, we consider it appropriate to strike a balance by averaging the solution so the result tends to be the most consistent shape between different views. Equation 13 models this loss, being M_P and M_T the rendered and ground truth masks.

$$L_M(M_P, M_T) = \frac{1}{|M_P|} \sum_{i=1}^{|M_P|} \left(M_{P_i} - M_{T_i}\right)^2 \tag{13}$$

With these losses, we can capture the desired shape and color, guiding the solution toward them. However, the solution must also present some desirable features, such as non-degeneration and smoothness. Therefore, to guide the reconstructions toward these desirable properties and to help narrow down the solution space, we define additional regularizations:

- Shape regularization. It favors a smooth mesh and avoids degeneration, being its use inspired by NVDiffRec. Equation 14 formulates it, where V is the set of vertices; F obtains the set of pairs of vertices that form a face with the input; and α_v is the angle at a vertex v in a given triangle. This regularization was designed by adapting the curvature flow smoothing presented in [57] because it allows a more uniform smoothing under uneven mesh distributions –as the mesh is optimized, we cannot grant a priori an even distribution–.

$$L_{CF}(V) = \sum_{v_i \in V} \left\| \sum_{v_j, v_k \in F(v_i)} (v_j - v_i)\cot \alpha_{v_k} + (v_k - v_i)\cot \alpha_{v_j} \right\|^2 \tag{14}$$

- Normal regularization. In our case, we estimate the normals automatically from the optimized vertices. Therefore, this loss favors meshes that induce automatic smooth normals. Equation 15 defines it based on Laplacian regularization defined in [52] and [57], which we apply over the normal vectors instead of vertices to penalize high differences between normals inside a neighborhood. In this equation, N and n_v are the one-ring neighborhood and normal of a vertex.

$$L_N(V) = \sum_{v_i \in V} \left\| \frac{1}{|N(v_i)|} \sum_{v_j \in N(v_i)} n_{v_j} - n_{v_i} \right\|^2 \tag{15}$$

- Color smoothness regularization. This loss favors color uniformity inside the same triangle. We accomplish this by choosing S samples at random each time we compute the function, estimating the average difference between the first and the rest of the samples. Through the optimization steps, this function favors uniformity between samples without prioritizing any given sample. Equation 16 defines it, where T is the set of triangles, C is the color vector, and $C^t_{R(s)}$ is the color of a randomly chosen sample in t. We set S to five in our system.

$$L_{CS}(T, C) = \sum_{t_i \in T} \left\| \frac{1}{S-1} \sum_{s=2}^{S} | C^{t_i}_{R(1)} - C^{t_i}_{R(s)} | \right\|^2 \tag{16}$$

- Spring regularization. Inspired by [30], it aims for a minimum solution by disfavoring overgrowing and balancing triangle sizes. Equation 17 defines this function, which measures the total edge length of the mesh.

$$L_S(T) = 0.0025 \sum_{t_i \in T} \sum_{v_1, v_2, v_3 \in t_i} \|v_1 - v_2\| + \|v_2 - v_3\| + \|v_3 - v_1\| \tag{17}$$

Finally, during our experimentations, we observed that shape and normal regularizations can still lead to overgrowing of the shape. However, shape regularization is crucial to avoid extreme degenerations as it controls the relative positions of the vertices, avoiding triangle interpenetrations. Therefore, we settled on balancing both regularizations by, on the one hand, applying decay to the normal regularization and a general low weight to the shape regularization to reduce the tendency to overgrow. In contrast, on the other hand, to keep control, avoid degenerations, and maintain higher

topology stability, especially in later stages, a progressive weight increase is applied to the shape regularization. We settled for a smooth linear increase in the shape weight while a fast logarithmic decrease for the normal, the latter to avoid falling into the overgrowing tendency. The total loss is expressed by Equations 18, 19, and 20, where MI is the maximum number of iterations and i is the current one.

$$L = 40L_M + 10L_C + 0.02d_{CF}(i)L_{CF} + 0.01d_N(i)L_N + 0.0002L_{CS} + L_S \tag{18}$$

$$d_{CF}(i) = 1 - \max\left(0.01, \left(1 - 1.5\frac{i}{MI}\right)\right) \tag{19}$$

$$d_N(i) = \max\left(0.05, \min\left(0.4, \left|\log\left(\frac{i}{MI} + 10^{-4}\right)^5\right|\right)\right) \tag{20}$$

4.3.3. Remeshing and resampling

With the aim of avoiding overgrowing of the reconstruction, we proposed the decay of the normal regularization while keeping the weight of the shape regularization generally low. Despite the increase in shape regularization weight during the process, this still leaves room for the appearance of slight shape degenerations. Taking inspiration from SFT, we apply periodic screened Poisson reconstruction to fix any possible shape artifact, followed by simplification to keep the number of faces constant.

However, this has the disadvantage of interfering with color estimation, as mesh colors are linked to triangle IDs, which are their respective index positions in the internal tensor. When remeshing, a new mesh is generated, redefining the vertices and the faces. As a result, triangles in the same spatial position generally have different IDs, shuffling colors along the surface. Therefore, part of the progress is lost, and colors must be refined again.

Instead of reinitializing the color as in [20], we propose a sampling method to recover lost progress. By storing a copy of the mesh before the remeshing, the colors of the new mesh can be updated by sampling it. From now on, we will refer to the input and output meshes of the remeshing as m^i and m^o, respectively. Similarly, the color vectors of the input and resulting meshes will be named $\vec{c}^i$ and $\vec{c}^o$. With this notation, the resampling procedure performs the following steps:

- For each triangle t_k^o in m^o, it computes the world coordinates s_n^o of every color sample inside t_k^o by using the barycentric coordinates to interpolate the triangle's vertex coordinates [81].
- For each triangle t_k^i in m^i, it computes the center $\bar{t}_k^i$ by averaging the positions of its vertices.
- The distance matrix from every color sample in m^o to each triangle center in m^i is computed. This allows finding the closest triangle of m^i to each sample in m^o, obtaining the set:

$$\mathscr{C} = \left\{ \left(s_n^o, t_k^i\right) \middle| s_n^o \in m^o, t_k^i = \arg\min_{t_h^i \in m^i} \|s_n^o - \bar{t}_h^i\| \right\} \tag{21}$$

- For every pair in $\mathscr{C}$, the barycentric coordinates of the projection of s_n^o into t_k^i can be obtained, following the proposal in [28]. With them, the index j in $\vec{c}^i$ of the projection is computed [81]. By

also obtaining the index h of s_n^o in $\vec{c}^o$, the color value $\vec{c}_j^i$ of the original mesh can be copied into $\vec{c}_h^o$ for the new mesh.

In summary, this approach finds the closest color in m^i for each sample in m^o. By assuming topological similarity between a mesh and its remeshed version, the colors for the new mesh are expected to be the same as the closest points in the original mesh. This is reasonable, as Poisson remeshing tends to generate a smoother version of the original mesh, preserving the initial topology unless very degenerated meshes are provided.

Our algorithm finds the closest triangle using the distance to its center. Even though this can fail in some cases, such as with very stretched triangles and samples near their extremes, it is generally a good approximation, allowing an efficient implementation with matrix operations. Additionally, it can be compensated by finding the N nearest triangles in m^i for each s_n^o, averaging the colors obtained for the N projections. This has a blurring effect, recovering a less detailed version of the original colors. However, this same effect also has the advantage of diminishing any possible error of the previous mesh, allowing for an easier correction by the optimization process. Nonetheless, the general colors are still restored, being able to take advantage of the previous progress to complete the color estimation further. We fixed the number of nearest triangles sampled to three.

4.3.4. Implementation

We implemented the system described so far by using Python and PyTorch. Therefore, all our internal data structures supporting the system are based on tensors. We leveraged the *pyredner* API as a differentiable rendering pipeline for our system, providing us with the required components and data structures to manage 3D scenes. This rendering pipeline was chosen because a differentiable ray tracing renderer can reveal artifacts hidden by local lighting, which helps to reduce ambiguity.

By default, the materials defined by *pyredner* are based on color constants or image textures as tensors. To use mesh colors on *pyredner*, Goel et al. modified the API to extend the materials to work correctly with one-dimensional tensors representing mesh colors. However, this was performed two years before our work and was never included in the official *pyredner* distribution.

Consequently, for using mesh colors in the current distribution of *pyredner*, we decided to modify the currently available source code distribution to include mesh colors, following the modifications performed by Goel et al. on the older version. We accomplished this through manual inspection and comparison between *pyredner*'s source files and the modified files available at Goel et al.'s repository, making the proper changes to add mesh color support. Most changes fell onto materials, their underlying texture structures, and their sampling, extending textures to support one-dimensional tensors accessed through the triangle IDs and barycentric coordinates. Once the *pyredner*'s source code modifications were performed, the library was compiled and installed using its default installation script *setup.py*.

Building upon these bases, our system follows design conventions inspired by NVDiffRec. The code *train.py* presents the optimization system, which parses the input, loads the proper resources, runs the optimization, and saves the result. The main arguments of the system are:

- *-i*, *—iterations*: total number of iterations for the optimization. In each iteration, all the reference images are used to optimize the mesh.

- —*mesh*: path to the initial mesh we wish to optimize, which is expected to be formatted as an OBJ file.
- —*texture-samples*: number of samples used to generate image textures from the optimized mesh colors.
- —*texture-refinement-steps*: number of iterations used to refine the image textures.
- —*batch-size*: we apply batch optimization. Therefore, in each iteration, the provided image set is divided into mutually exclusive subsets of size equal to the batch size, being the parameters updated once per subset.
- —*remeshing-interval*: it indicates the remeshing period. Given a value *n*, the remeshing operation will be applied every *n* iterations.
- —*longtail*: it defines the number of iterations dedicated to joint geometry and material optimization. Once this number of iterations is reached, the rest of the process focuses on color refinement.
- —*views*: path to the JSON file containing matrices defining the camera rotation around the origin of coordinates for each view.
- —*cameras*: path to the JSON file containing the camera positions and up vectors for each image.

Once processed the arguments, the reference images and cameras are loaded as tensors. The images are expected to be RGBA, defining their mask on the alpha channel. For the cameras, it is always assumed that their looking direction is toward the origin of the scene. When the —*views* argument is used, the position and up vector of each camera are computed from the matrix, fixing the camera at a distance of two units from the origin. When —*cameras* is used, the positions and up vectors are directly employed to set up the *pyredner.Camera* objects for each image. Cameras in the former case are assumed to be orthographic, while cameras in the second case can be either perspective or orthographic, determined by their JSON configuration.

Next, the initial mesh is loaded using utils provided by *pyredner*. The result is a *pyredner.Shape* object whose *vertices* tensor property is prepared to be optimized. Additionally, a new *pyredner.Material* object for the mesh is created, using only a diffuse component defined from a uni-dimensional mesh colors texture, initialized randomly. The underlying tensor for the material is also prepared to be optimized. With the required components, a list of *pyredner.Scene* objects can be built for each reference image, containing the reference to the proper camera, shape, material, and a fixed common white environment, also previously loaded.

While *train.py* manages the optimization loop, running it for the number of specified iterations, our defined class *Optimizer* executes the optimization. Given the shapes, materials, target images, and configurations, *Optimizer* sets up the Adam optimizer and our *Remesher* class. Then, an optimization iteration can be applied by running its *step* function. Internally, this method performs an optimization step for each batch, computing the losses depending on the training status and properly updating the parameters. Additionally, once finished the iteration, it manages the use of remeshing to repair the mesh if appropriate.

Remeshing is performed through our *Remesher* class. This class encapsulates the remeshing process by using the screened Poisson and simplification operations provided by *pymeshlab* [53], applying our proposed resampling over the result. Therefore, after every remeshing step, the optimized mesh is replaced by the newly generated mesh.

As stated at the beginning of Section 4.3, we employ a long-tail optimization approach inspired by NVDiffRec. This consists of optimizing the shape and colors jointly until a given iteration later in the process, when the geometry is fixed, and only the color continues to be optimized. This aims to compensate for the interference of the remeshing step in the detail of the color estimation and to allow a better color refinement once all vertices remain still. Given the initial configuration, *Optimizer* automatically applies this scheme. If the remeshing interval is not zero, remeshing is also used right before fixing the geometry.

Once the optimization is complete, the results are exported. To keep compatibility with external tools, the estimated color for the mesh needs to be converted into a 2D texture format. To do this, we rely on a second optimization phase inspired by NVDiffRec. This phase is targeted to generate image textures from the current optimization. Therefore, high-quality renders from random viewpoints around the mesh are generated as reference samples from the reconstructed mesh. Then, based on these samples, a random image texture is linked to the fixed mesh and optimized to reproduce the same appearance as the mesh colors. This optimization uses a simplified operation of *Optimizer* without remeshing or long-tail, using as loss functions the color loss in Equation 12 and the texture smoothness regularization in [52], being their weights 10 and 0.002 respectively. The number of optimization steps for the image texture is given by *—texture-refinement-steps*.

Before closing the section, making some final remarks about the implementation is important. Early in the development phase, we considered optimizing a tetrahedral grid by using the Differentiable Marching Tetrahedra similarly to NVDiffRec, leveraging their *DMTet* implementation. However, it was observed that this kind of representation tends to generate holed meshes, being difficult to obtain a uniform shape, as it will be further shown in Section 5.1. Therefore, we discarded this alternative to use triangular meshes (as Goel et al. [20] did) and regularizations.

Additionally, it was observed that the automatic generation of normals for the mesh after each update using the tools provided by *pyredner* could lead to undesired cases of inverted normals. This situation blocked the color optimization, as the surfaces were rendered as purely black. To detect this situation and reverse the generated normals, a naive approach of generating a low-resolution render of the mesh with purely white material and measuring the level of black was used.

As we have seen, our system requires the initial mesh to be optimized. This was done to increase modularity and make the optimization procedure independent from the initial mesh estimation. Given a subset of reference samples and viewpoints, the initial mesh can be generated following our proposed strategy through our script *build_initial_mesh_v3.py*. This script takes as inputs the JSON file describing the images and view rotations (*—cameras*), the desired result filename (*—output*), and the scale-down factor for the source images (*—reduction*). From them, *numpy* vectors are used to define the voxel projection for each image and obtain the final voxel representation, which is converted into a mesh using the *mcubes* API [91]. Post-processing is applied to scale down the model using the images as references, as well as to remesh and simplify the mesh using *pymeshlab*.

Lastly, to increase the robustness of the system, we implemented a custom checkpoint mechanism. Said system, encapsulated by the classes *CheckpointManager* and *TrainingStatus*, saves the optimized mesh and color vector at fixed intervals during the optimization, as well as a status file. Then, based on this file, automatic recovery of the last checkpoint is enabled when relaunching the optimization, helping to recover the progress in case of any failure.

When considering our proposal, some configurations are fixed in our experiments unless otherwise stated. A batch size of four samples is used, and the initial mesh is estimated from a frontal and a side sample. Additionally, optimization renders use one bounce and four samples. Finally, ten

reference views, 100 steps, and a texture resolution of 2048 by 2048 pixels are used for image texture generation. We use a learning rate of 0.005 for the reconstruction and 0.05 for the texture generation.

5. Results

Given our two proposals, this section will be structured into two main subsections. Firstly, we will present the experimentation carried out with NVDiffRec over the domain of non-realistic depictions. The objective of this section will be to help determine the viability of this system and our suggested workflow for the task, studying the degree of generality of NVDiffRec. For this, we will present several use cases over which we will observe NVDiffRec's performance.

Secondly, we will present the results obtained with our proposed system. We will introduce the datasets used for its evaluation, perform the ablation study of our contributions, and compare the system with NVDiffRec and SFT. This later comparison will aim to determine if our proposal adapts better to sketches compared to more generic approaches designed for realistic images. Additionally, this section will also perform a controlled study on how different quality factors in sketches affect the reconstruction, which will be key for determining our system's weaknesses and where future research efforts should be led.

5.1. First proposal

Our first proposal had a dual intent. On the one hand, by using the state of the art in inverse rendering reconstruction, we aimed to determine the adequacy of this method for its application over illustrations. On the other hand, we wanted to determine if the promising NVDiffRec was general enough to be applied to our domain successfully. Therefore, the experiments in this section were designed to this extent. We experimented with NVDiffRec over three use cases.

5.1.1. Use cases

Our use cases represent three situations of interest that allowed us to apply NVDiffRec under different conditions. In essence, these studies accomplished the purpose of using the system and the proposed workflow, allowing us to better determine their usability and appropriateness for our intended task. These experiments were performed before our second proposal as an initial proof of concept, and their results helped us to know how to guide and continue with our research. Therefore, these use cases do not aim to test NVDiffRec over drawings exhaustively but instead allow a general idea of its capabilities and usability. A more formal evaluation will be carried out in our comparison segment in Section 5.2.

Sphere. As our first use case, we introduce a simpler instance to observe NVDiffRec's base results under the sketches domain. We present a digitally drawn circle, already masked and shown in Figure 24. By repeatedly providing this image from multiple views around the scene's center at a fixed distance, we aimed to simulate the multi-view samples of a sphere. Two approaches were used to generate the view matrices:

- Simulating a turn-around of 28 frames around the vertical axis.
- Generating completely random rotations around the scene's center at a fixed distance. This method implies that samples are generated dynamically during the optimization, providing a new random viewpoint each time.

This use case is inherently simple by nature. However, it has the advantage of presenting highly consistent samples, allowing a custom number of reference samples, and being the black outline

line the major inconsistency between views. Therefore, this allows observing the results obtained under a reasonable base case.

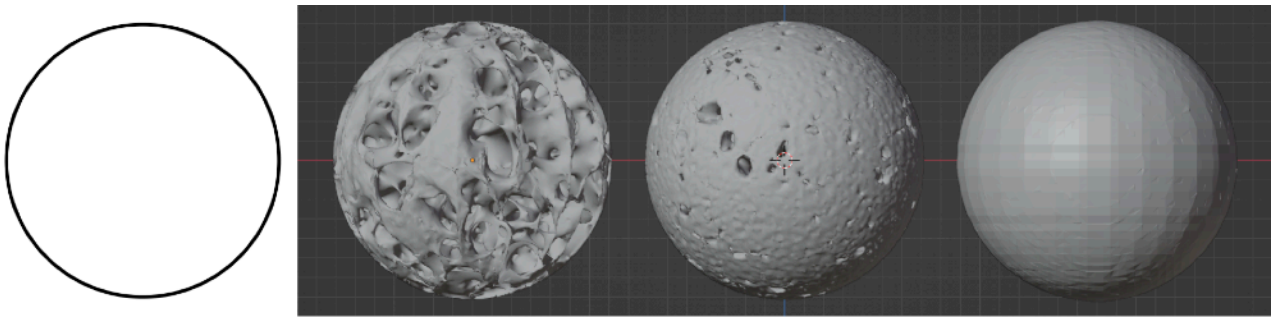

Figure 24. Reconstruction results for the drawn sphere.

Left to right, ground truth, estimation by NVDiffRec with turn-around, estimation with random rotations, and estimation with Visual Hull [34, 94] with turn-around.

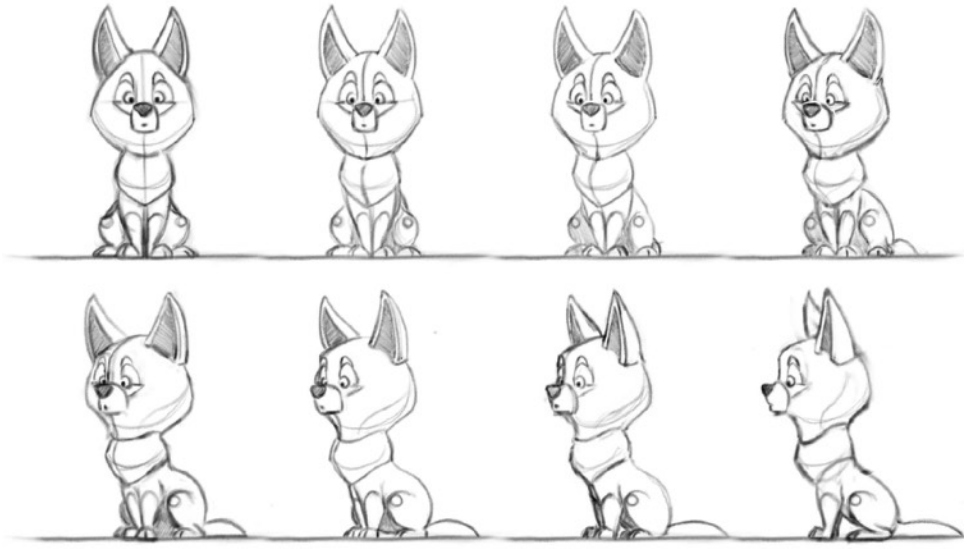

Figure 25. Sketches depicting the partial turn-around of a fictional dog.

The complete set of frames, created by Anja Regnery, can be found in [62], containing 28 reference views.

Dog sketches. In this use case, sketches of a dog like the ones in Figure 25 are available, corresponding with the 28 frames of a complete turn-around animation. This use case was kindly provided by Anja Regnery [62], and it represents a real hand-drawn set of sketches depicting a fictional dog character from multiple views. Therefore, the benefit of this example is that it fully represents our task, having the advantage of presenting a wide set of samples to use reconstruction.

To use the dog sketches with NVDiffRec, we generated their masks and the camera poses for each frame. Due to the reduced number of images, we opted for generating the masks manually using Gimp, alpha masking outside the black outline, and erasing the ground line. This provided us with higher precision masks. In contrast, the view matrices for each frame were estimated thanks to the turn-around nature of the source, following the strategy we have detailed in Sections 4.2.2 and 4.2.4. As black-and-white lined sketches are simple in nature, they contain a limited number of key points. This heavily limits the applicability of COLMAP to this kind of sketch, usually failing in this case. Consequently, automatic viewpoint generation has to be reserved for more detailed cases.

Game character. The last use case we propose consists of reconstructing the character of a third-person view game in which the camera can freely move around it. However, some remarks must be made about the election of this use case.

The reason for this example resides in the non-realistic-looking nature of the content –like a painting– and the ease of generating samples. As previously mentioned in Section 4.1, by considering the color of the drawings, we extend our scope to more completed and detailed types of illustrations. This use case helps us simulate a more complete scenario than sketches, allowing us to explore our automatic workflow better. Nonetheless, its similarity to actual drawings is only partly due to its high geometric consistency through views given its synthetic origin. Additionally, it allowed us to obtain many samples, which, even though it enables dataset sizes closer to what was initially intended in NVDiffRec, is unfeasible with real drawings. Despite these issues, we still consider it a practical example as it allowed us to deal with challenging automatic mask generation and view estimation, studying their effects on the reconstruction.

For this use case, we took a screen recording of a game as a sample source, depicting the camera moving around the standing character. By extracting all the video frames, we obtained a total of 921 multi-view images of said character.

Figure 26. Samples corresponding to the game character use case.

From top to bottom, the original samples, the samples masked automatically with PointRend, and the samples with improved masks using a bounding box.

Given the nature of the capture method used to generate the samples, there is high variability in their contents. Moreover, as the camera was controlled manually during the recording, we cannot assume uniformity in its movements. Furthermore, the camera's movement is random and cannot be

either assumed. Therefore, for obtaining the masks and camera poses of these samples, we relied on the automatic approaches through Detectron2 and COLMAP, respectively detailed in Sections 4.2.1 and 4.2.2. Figure 26 shows some examples of the resulting samples.

5.1.2. Results

In this section, we report the reconstruction results for each one of the use cases presented in Section 5.1.1. Our evaluation of these results will be mainly visual, relying on image metrics and manual inspection. While the most challenging cases are the dog sketches and the game character use cases, a reference 3D shape does not exist for the former, and the character model for the latter is not openly available.

It is important to note that, for all the experiments detailed with NVDiffRec, we used 5000 iterations, random initial textures, texture resolution of 1024 by 1024 pixels, batch size of four, grid resolution of 128, and reconstruction in two phases with learning rates of 0.03 and 0.003. When COLMAP estimation was needed, we used all the full-resolution images available without masking. Shared parameters were used for the cameras and default configuration for the remaining attributes.

Sphere. In this case, we used orthographic cameras, matching the nature of the reference sample. Figure 24 shows the meshes obtained with NVDiffRec when estimating the simulated sphere. For the sake of comparison, the same estimation obtained through a more classic approach, the Visual Hull algorithm, is also provided. The Visual Hull reconstruction was generated thanks to the algorithm implementation in [34, 94] by providing the masks of the 28 images of the turn-around setup and their associated camera views, obtaining the intersection hull of all projections.

The results obtained with this toy example already reveal a tendency that will be recurring throughout this work when dealing with NVDiffRec. As can be seen, the reconstructions generated by NVDiffRec closely match the sphere silhouette given the reference viewpoints. However, especially in the turn-around case, we observe many holes and strange topologies for the surface, being the 28 segments “engraved” into the shape. This is because the lack of shading on the sketches relegates all the geometry information to the silhouette. Additionally, as NVDiffRec uses local lighting, there is no self-shadowing when rendering the mesh, not revealing the holes in the renders. Consequently, as holes do not generate feedback, the only feedback regarding shape is the silhouette. This can also be seen in the improvement with random views compared to the turn-around. As the number of silhouettes seen from different viewpoints is higher and more varied, the shape is more closely approximated. Nonetheless, in this simple case, Visual Hull offers a far better reconstruction alternative as it does not produce holes and closely approximates a perfect sphere.

Dog sketches. Several tests were performed with this use case. The first experiment was executed using a perspective camera projection, environment light optimization, and a training resolution of 550 by 550 pixels. All 28 sketches were used for training. The progress result saved by NVDiffRec during the last iteration can be seen in Figure 27.

The same tendencies as with the sphere can be observed in this case. NVDiffRec tries to approximate the silhouette of the dog and the general shape obtained when rendering from the given viewpoints. However, the lower parts of the body, such as the tail and paws, are not recovered. This can be attributed to the inconsistency between the projection in the sketches and the perspective camera, as the extremities –further from the center– are the most affected areas by the perspective deformation. As reference sketches tend to avoid perspective deformation, they are usually more closely explained by an orthographic projection. Furthermore, we can also appreciate how the grey

lines are reproduced using the lighting instead of the textures. Effectively, this is also caused by the lack of shading in the sketches, which increases the ambiguity regarding the effects of illumination and materials.

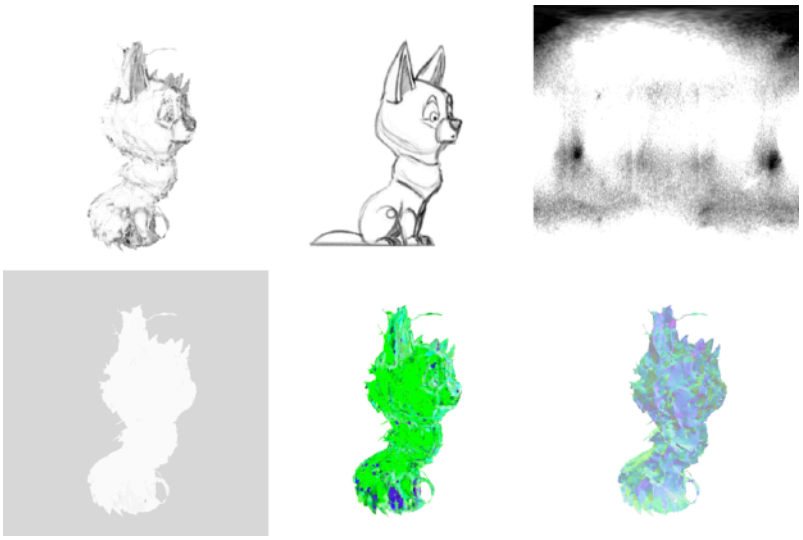

Figure 27. Results saved on the last training iteration in NVDiffRec (initial experiment).

From top to bottom, left to right, rendered mesh, ground truth, environment map, diffuse texture, specular texture, and normal map. A grey background has been added to the diffuse map to increase its visibility.

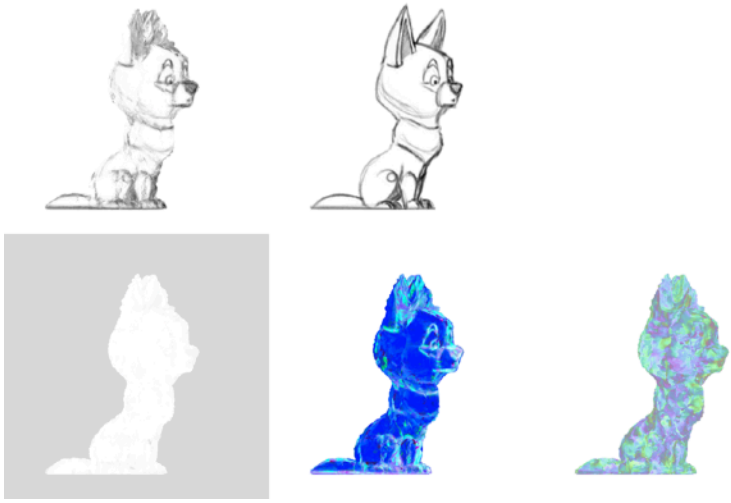

Figure 28. Results saved on the last training iteration in NVDiffRec (second experiment).

From top to bottom, left to right, rendered mesh, ground truth, environment map, diffuse texture, specular texture, and normal map. The environment map is a fixed pure white color, while a grey background has been added to the diffuse map to increase its visibility.

Therefore, we repeated the same training using an orthographic projection and fixed white environment light to improve the reconstruction. The result can be found in Figure 28. It can be observed that orthographic projection allows for a closer matching of silhouettes and outlines between reconstruction and sketches. Moreover, the tail and paws are now recovered thanks to the lack of perspective distortion.

However, we can still observe how the reconstruction presents very sharp surface features and holes, having a similar defect to the sphere turn-around where the segments are integrated as surface

features. Even though the silhouettes are detailed from the reference viewpoints, the general topology and views from other angles do not match the expected shape when considering a dog. This phenomenon has two main causes. On the one hand, the SDF-based representation discretizes the appearance of geometry, only generating new faces when the values change sign, easing the modeling of holes. On the other hand, as discussed previously, the source's lack of shading and the use of local lighting hide these holes from the optimization, avoiding their refinement. We will refer to this reconstruction phenomenon as the “shadow puzzle” effect. We can also see that the sketch lines were integrated into the material this time but on the specular map. This shows that there is still ambiguity regarding material behavior due to the lack of shading.

Finally, we increased the number of samples trying to add additional silhouettes to improve the reconstruction inspired by the results over the sphere. We accomplished this by generating two additional frames between the existing ones thanks to AnimeInterp [43], a frame interpolation system specially designed to work over cartoon animations. With this technique, the number of samples was increased to 84, and the experiment was repeated using all of them. Figure 29 shows a comparison of the meshes obtained with each experiment. Again, the reconstruction obtained via Visual Hull [34, 94] with the set of 28 samples has also been added for reference.

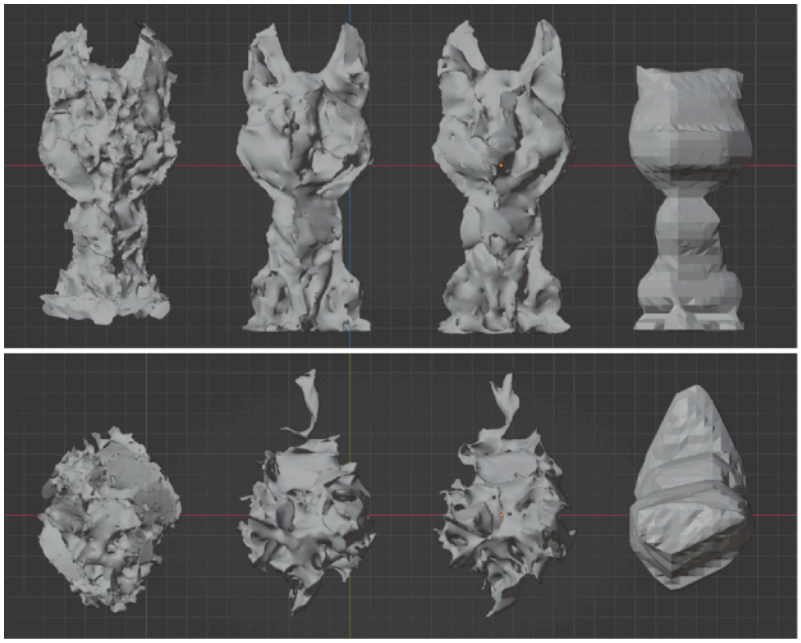

Figure 29. Reconstructed meshes for the dog from the front and top.

Left to right, perspective, orthographic, orthographic interpolated, and Visual Hull estimations.

As we can see from the comparison of the results, the shadow puzzle effect is still significantly present despite increasing the number of available samples. However, we can see a slight increase in silhouette definition and quality. We argue that this points to the fact that more side views are not necessarily required. Instead, top and bottom silhouettes would be necessary to further reduce the appearance of holes, reproducing the results seen in the sphere. Unfortunately, these cases cannot be explored as top and bottom sketches for the dog are not available.

However, it is relevant to note that, in this case, the silhouettes of the reconstructions provided by NVDiffRec are more detailed and recognizable than the reconstruction provided by Visual Hull. This shows that the flexibility offered by NVDiffRec due to its optimization nature can surpass the fitting capability of the more rigid nature of Visual Hull under challenging scenarios with

inconsistencies. Therefore, the application of inverse rendering optimization approaches is still relevant.

Game character. A total of 921 images of 1920 by 1342 pixels were obtained by extracting all the frames from the source video. While all of them were used for the COLAMP estimation, 737 were employed to build the training split, and 184 formed the validation split. These splits were automatically masked, removing from the resulting sets the empty images and resizing every sample to half size.

We divided the experiments into two groups to study the effects of the center estimation and the masks. On the one hand, we performed experiments in which the different strategies for center estimation were applied with automatic masks. On the other hand, experiments with improved masks were carried out for comparison. In all cases, perspective cameras were used to match the ground truth images, the training resolution was 960 by 671 pixels, and the lighting was learned.

Estimation	MSE ↓	PSNR ↑
GRASP	0.008	23.77
Bounding box	0.008	23.90
Averaged	0.008	**23.93**

Table 3. Validation metrics for the game character reconstruction (initial experiments).

The average MSE and PSNR were obtained from 132 validation samples with reconstructions of the game character for different center estimations with automatic masks.

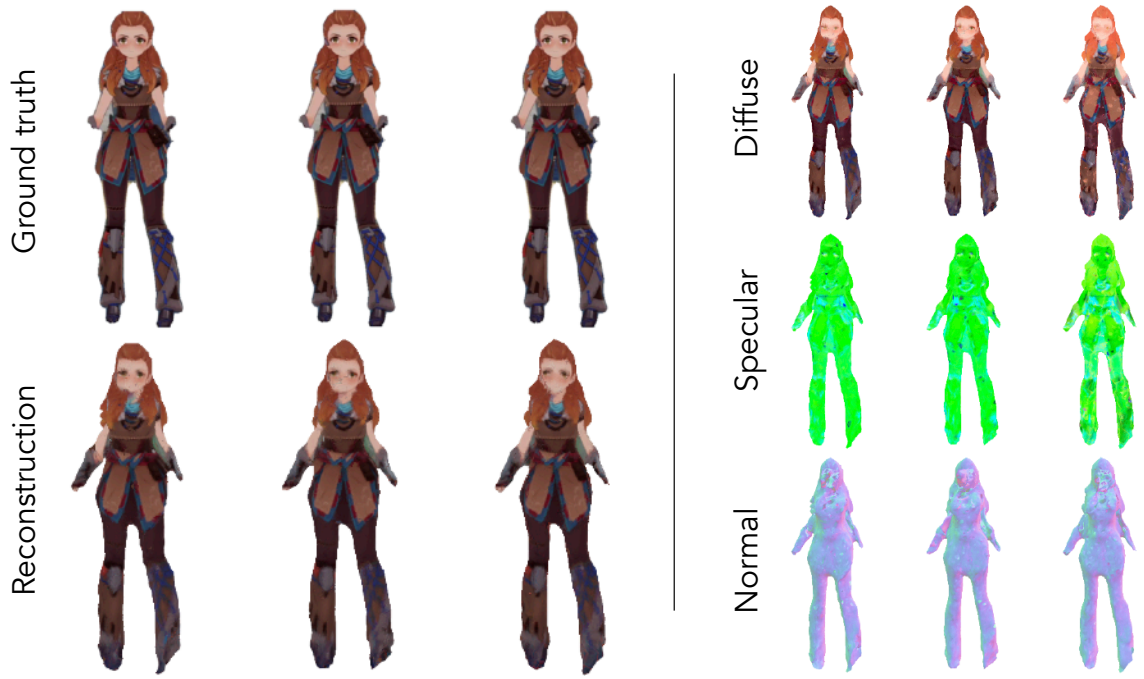

Figure 30. Game character reconstructions with maps (initial experiments).

The reconstructions for different center estimations on the game character are compared with the first validation sample. In each set, left to right, GRASP, BB, and averaged. Images have been cropped for visibility.

Firstly, we applied independent NVDiffRec optimizations with the same images for views estimated with different center estimations, namely GRAPS, BB projection, and averaging both. Table 3 presents the numeric results provided by NVDiffRec in validation, reporting the image metrics of the reconstruction renders compared with the ground truth. Figures 30 and 31 visually show the obtained reconstruction. It is important to note that, after the masking and filtering, the dataset for these experiments was reduced to 517 samples for training and 132 for validation.

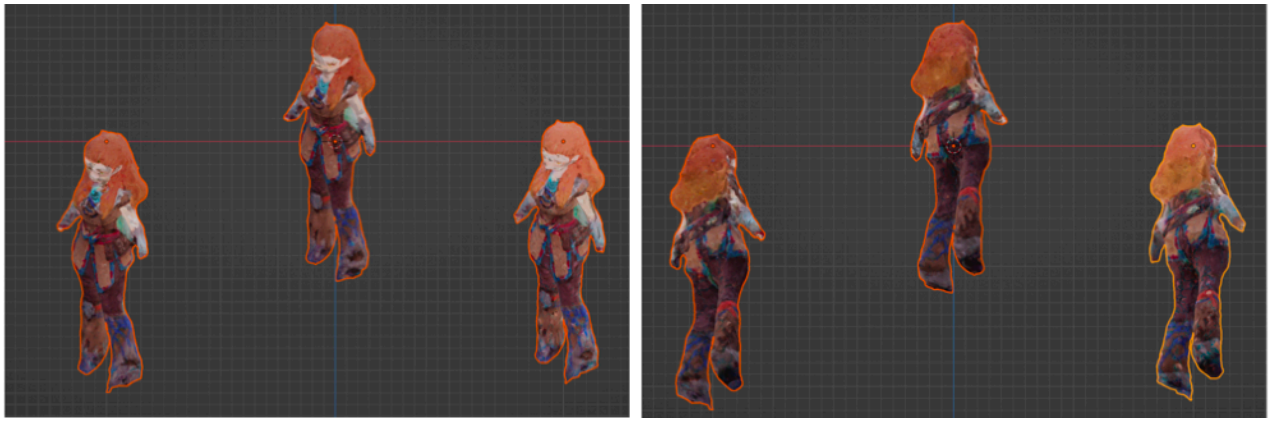

Figure 31. Generated 3D models for the game character (initial experiments).

Both frontal and back views are displayed. Left to right in each set, GRASP, BB, and averaged estimations.

We can see that the results are similar for all center estimation techniques. Looking at Table 3, the averaged and BB estimations obtain slightly better PSNR, although the MSE is similar in all cases. Figures 30 and 31 show that the differences are found in details like the hair shaping and the surface texture. This is reasonable as the center difference is relatively small and mostly displaced in the vertical axis. Therefore, given that the camera estimation by COLMAP is consistent enough, the reconstructions are robust to this vertical displacement of the center as we are still located around the tetrahedral grid. However, the difference in the estimation can be clearly seen in Figure 31, as the center of the resulting meshes is different.

It is worth noting that all cases fail to recover the hands, and the feet are very roughly reconstructed. On the one hand, the former is because the hands are usually not correctly included in the automatic masks, being mainly classified as not part of the target. On the other hand, the feet are generally well captured, but they are small, being rounded off and merging with the thicker part of the boots during reconstruction. We can also observe that the models have an implicit rotation. This is due to the COLAMP estimation of the system of coordinates. Even though we displaced it, we did not modify its imposed orientation, causing the whole scene to be rotated. However, this is not a critical issue as it only affects the result's rotation and does not interfere with its reconstruction, being easily rectifiable over the obtained mesh.

Secondly, we used the best center estimation techniques to evaluate the effects of the masks on the result. Given that we had assumed the availability of the BB containing the target, we used this information to improve the masks. As reported in Section 4.2.4, we did this by automatically masking anything located outside the bounding box and filtering out those samples whose bounding box interior was empty. This method allowed for more refined masks, obtaining 501 training and 124 evaluation samples.

We again tested the reconstruction for the BB and the averaged origin estimations but this time with the improved masks. Table 4 shows the metrics obtained in validation for all models over the

validation set with improved masks. Note that the enhanced masks only help reduce the noise but do not improve the character's silhouette. Figure 32 shows a visual comparison of the estimated 3D models.

Estimation	MSE ↓	PSNR ↑
Bounding box	0.004	25.03
Averaged	0.006	23.85
Bounding box⁺	**0.003**	27.73
Averaged⁺	**0.003**	**27.77**

Table 4. Validation metrics for the game character reconstruction (mask experiments).

The average MSE and PSNR were obtained from 124 improved validation samples with reconstructions of the game character with automatic and improved masks.

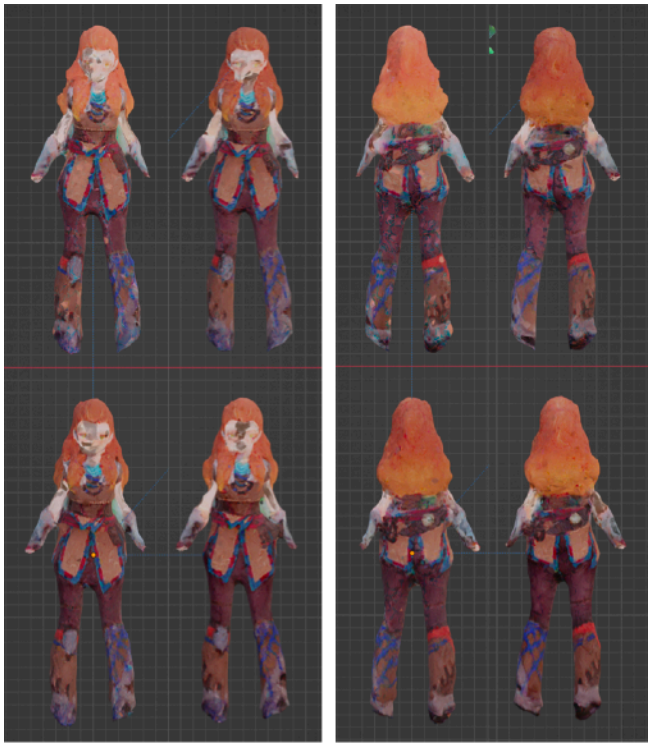

Figure 32. Models obtained for the game character without and with improved masks.

Both frontal and back views are shown. In each set, left to right, estimation with normal and improved masks, top to bottom, average, and BB estimations. All models have been rotated for the comparison; the original orientations were as in Figure 31.

The reconstructions with the improved masks are visually similar to the initially obtained ones. Looking at Figure 32, we can see a slight increase in the sharpness of the textures. Regarding the 3D shape, we can observe small improvements in the shaping of the hair. In Table 4, we can see that the models trained with improved masks perform numerically better than those trained with fully automatic masks.

These results are reasonable because, as we said previously, the masks improved by the bounding box only help reduce the noisy parts outside it and better detect the samples where the character is erroneously masked out. Given that the amount of filtered images is low, reducing the set from 517

training images to 501, the improvement on the reconstruction can only be minor in this particular case. Moreover, filtering noise outside the bounding box mainly helps at a geometric level as the generation of far-floating geometry is avoided, which is reflected in the results with averaged estimation. However, given the simplicity of the approach, we still consider that its improvements are beneficial as they help reduce the geometric and color noises in the reconstruction.

5.2. Second proposal

Our second proposal aimed to adapt existing inverse rendering techniques for reconstruction over realistic images to increase their suitability for sketches and flat-colored drawings. Therefore, our objective through this section will be double. First, we will evaluate our system to determine the effect and relevance of our proposed modifications, analyzing how the system reacts under different challenges commonly present in sketches. Second, we will study how the results of our tailored proposal compare to those obtainable with standard approaches not adapted to the domain, exploring the relevancy and effectiveness of our proposal.

5.2.1. Datasets

With the aim of higher control over the datasets, allowing us to perform a detailed study of our proposal, we opted to test our system over two synthetic examples. Even though synthetic sketches overlook the subjectivity of hand-drawn samples, not fully representing our task, they are a baseline. When working with samples without subjectivity interference, we work under the best case possible for our system, and the results obtained should be evaluated accordingly. Additionally, using synthetic sketches allowed us to alter their quality freely, enabling the study of different input factors.

We gathered two existing 3D models to generate sketch-like reference samples: Axolotl [13] and Vasque [14], processing them to fit a cube of two units. These models present different desirable features for our evaluations, being both of medium complexity. While Axolotl presents multiple colors, roundness, and asymmetries, Vasque presents sharp edges, concaveness, and symmetry around the vertical axis. Therefore, these models are representative of different frequent features in common objects.

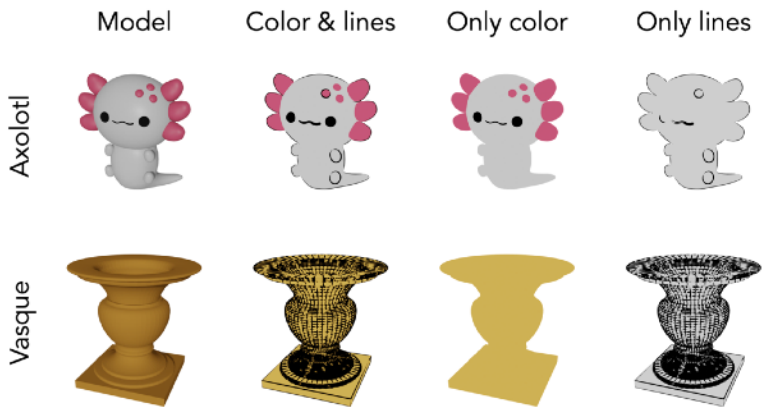

Figure 33. Models used and synthetic sketches in three styles.

For generating synthetic renders reproducing a sketched appearance, we used Blender and its Freestyle module as rendering pipelines due to their flexibility, familiarity of use, and powerful automatization through Python scripting. We generated synthetic samples for each model in three

styles representative of the most common use cases: lined sketches without color, flat-colored lined sketches, and flat-colored sketches without lines, shown in Figure 33. Each generated set contained 128 training and 128 validation random view samples of 512 by 512 pixels. The views were distributed on the surface of a sphere of radius five around the center of the scene, always looking at it. From now on, we will refer to the flat-colored lined sketches as the reference set.

We also generated modifications of the initial datasets to study different quality factors. This was done in all cases by modifying the generation pipeline to alter the quality attributes of the multi-view samples. In each modified dataset, only the property under study was changed, keeping the rest of the generation parameters as the reference set for comparison. However, it should be noted that the views between different datasets may differ due to the random nature of the view generation. This is compensated by the fact that we will be using a completely random generation, and a high number of views is mainly considered, being able to assume a uniform distribution of viewpoints and, therefore, negligible effects over the reconstruction.

Finally, to complete the evaluation of our system, we applied it to real hand-drawn sketches. However, as mentioned in Section 2.1, the available paired datasets containing both sketches and 3D models are very limited, even more when considering the multi-view case. During this work's development, we found two publicly available datasets providing hand-drawn sketches and 3D models.

On the one hand, Xiao et al. [79] proposed a dataset of multi-view sketches corresponding to 3D models of different categories drawn by people with varying skill levels, containing 3620 sketches. Through their work, they presented a quantitative analysis of the difference between professional and novice sketches and how the proposed dataset enabled synthesis and reconstruction-related tasks. Despite the desirable features of this dataset, such as the explicit multi-view of the sketches, their hand-drawn nature, and the availability of models, it lacked relevant information required for our system. Even though a multi-view setup was provided, the camera poses used for the views were not included in the set. Moreover, given the simplicity of the renders and sketches, it was not feasible to use COLMAP for their estimation. Therefore, using [79] in our system was not viable.

On the other hand, Gryaditskaya et al. [23] presented a work analyzing the properties and features of technical sketches and the taxonomy of the lines involved in their design. As part of their research, they generated a dataset of hand-drawn technical sketches from 3D models called OpenSketch. Even though this dataset was not explicitly multi-view, each participant designed the given object sketches from a different view, allowing to obtain multi-view samples by collecting the drawings made by various participants. Furthermore, this dataset, although smaller than [79] with 180 sketches, included information such as the camera estimation for each view, vectorial representations of the sketches, time stamps, and labels for the lines of a subset of 107 drawings. Despite all the available information in OpenSketch, its use in our system proved challenging.

Firstly, as far as we could study, the camera poses provided did not allow for a direct correspondence between sketches and 3D models, as the drawings were not centered. This made the matching between renders and illustrations inappropriate for reconstruction through our system by default. Moreover, the sketches lacked masks, and the more abstract nature of the depicted objects, joined with the disconnectedness between sketch lines, made it difficult to obtain them automatically. This showcases the high requirements that the inverse rendering approach imposes on the input data, which we will discuss in Chapter 6. Given the considerable amount of difficulties that the use of OpenSketch presented and the high deviation of focus that would have implied

overcoming them, we decided to discard the use of this dataset to concentrate our efforts on the evaluation of our system.

Therefore, to study the system over real scenarios, we designed a custom dataset. In particular, we manually drew four canonical views for four different imaginary objects, namely Plane, Shrimp, Vase, and Abstract, scanning and masking them. The viewpoints were manually defined thanks to the canonical nature, meaning we used views related to the object's elevation, plant, and profile. To compare the results against a 3D model, we manually created low poly 3D objects from those sketches. This will allow us to measure the system reconstruction against the human-made 3D interpretation of the sketches.

5.2.2. Baseline results

As our first experiment, we performed two reconstructions for each initial training set: one using remeshing and one without it. This allowed us to obtain the base metrics for our system that will be the point of comparison throughout the ablation and quality factors study. In all cases, 30+3 iterations were used, meaning the last three only refined color following our long-tail scheme. When remeshing was used, it was applied every two iterations. We will refer to this configuration as the reference configuration from now on.

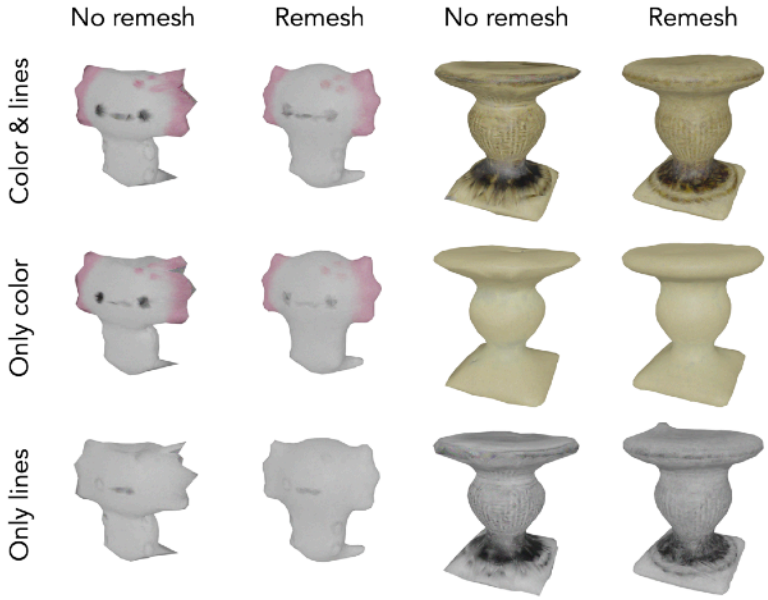

Figure 34. Baseline results on different styles.

We evaluated the reconstructions in the model and image spaces using the generated validation images and the original models. For the image domain, we measured the Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), Structural similarity (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS). The latter was measured based on Visual Geometry Group (VGG), using the function provided by *torchmetrics* [90]. Meanwhile, the Chamfer distance was measured to evaluate the similarity between models in 3D space. To compensate for possible scale mismatches between the compared objects, we measured the pure Chamfer distance and the distance after scaling the reconstruction to fit the largest dimension of the reference model. In this way, cases with different scales but similar topologies are not penalized.

Figure 34 and Table 5 show that the results present an acceptable quality. Even though sharpness in silhouettes and colors has been lost, the figures are easily recognizable. We can see how our color loss effectively preserves consistent lines between views while discarding the rest. Nonetheless, better color estimation is still observed for the style without lines, as it is the most consistent. We can see how the concaveness in Vasque is lost, presenting a flat top. This is reasonable considering the lack of volume shading in the reference samples and the regularizations we defined for shape and normals, guiding the optimization towards smoother and uniform solutions. Finally, we can observe how remeshing works best for Axololt due to its more rounded nature than Vasque. This tendency will be consistent throughout our experiments, pointing to the fact that the use of remeshing should be decided on a case-per-case basis. While it can correct mesh artifacts, it also makes it challenging to recover sharper shapes, being more appropriate for highly inconsistent cases or rounder objects.

Model	Axolotl						Vasque					
Style	Color + lines		Only color		Only lines		Color + lines		Only color		Only lines	
Remesh.	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
MSE ↓	**0.007**	**0.007**	0.005	**0.004**	**0.007**	0.006	0.026	0.027	**0.009**	0.010	**0.033**	0.031
PSNR ↑	**21.62**	21.79	23.30	**23.99**	21.76	22.23	15.91	15.69	**20.57**	20.06	**14.91**	15.11
SSIM ↑	**0.923**	0.926	0.934	**0.939**	0.925	0.929	0.792	**0.789**	**0.871**	0.870	0.790	**0.789**
LPIPS ↓	0.109	**0.107**	0.140	0.139	0.147	**0.149**	**0.185**	0.197	0.191	0.190	0.246	**0.247**
Chamfer ↓	0.057	**0.083**	**0.056**	0.082	0.057	0.081	0.079	0.113	0.076	0.108	**0.074**	**0.119**
Scaled Chamfer ↓	**0.014**	**0.008**	0.013	**0.008**	0.013	**0.008**	0.025	0.034	0.023	0.030	**0.020**	**0.037**

Table 5. Validation metrics of the reconstructions from different styles.

The metrics were obtained from 128 validation samples with reconstructions over 128 training samples in each style. The best and worst values are depicted in green and red, respectively.

5.2.3. Quality factors study

Drawings contain noise and inconsistencies. Even for highly skilled artists, keeping an object's geometry and appearance constant between multiple views is challenging. These aspects need to be considered when designing a reconstruction system from sketches. Aiming to study how our system performs under less ideal conditions, we simulated the most frequent quality factors regarding drawings to see how they affect the results compared to the baseline.

When considering reconstruction from images in any domain, the most common factors are the number of available samples, their resolution, and the precision of their masks when required. However, more specific to the sketched domain, we also must consider the consistency between viewpoints and views, and the consistency of the geometry between views. While the former relates to the possible discrepancies between the “real” viewpoints of the images and the ones provided to the system, the latter refers to potential shape inconsistencies between views due to either precision

errors or stylistic choices. To study these factors, we present modifications of the initial sets whose reconstruction will allow us to observe their effects on the result.

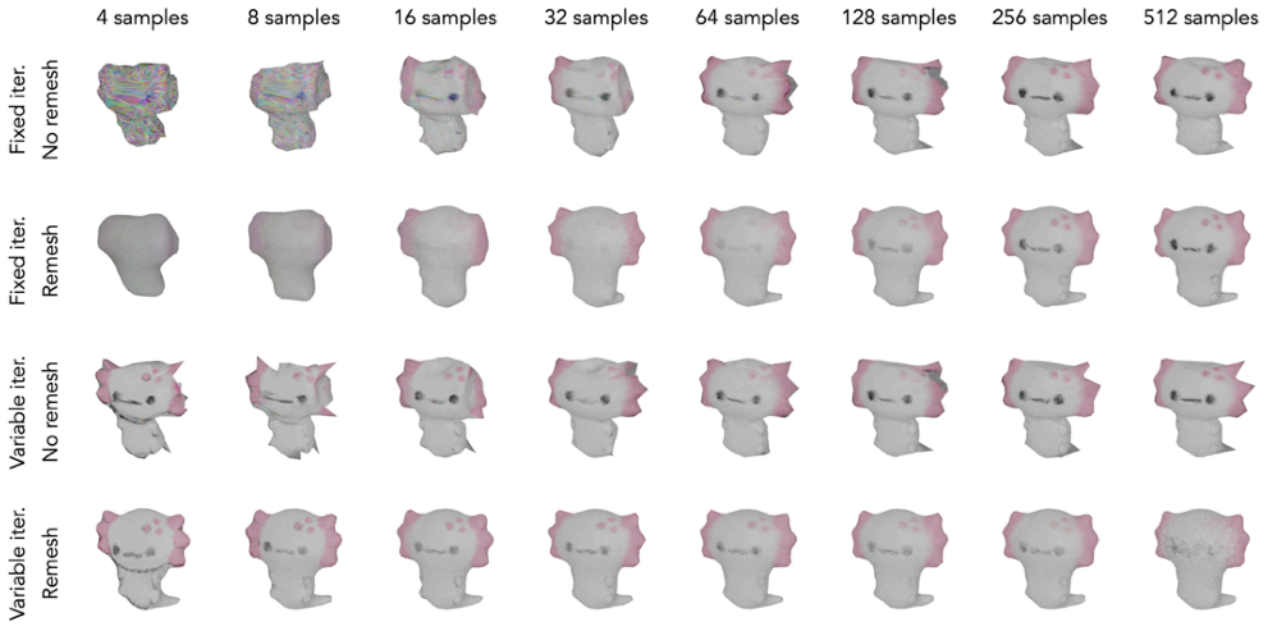

Figure 35. Reconstruction results over Axolotl with an increasing number of samples.

Firstly, we evaluated the effect of the number of reference images in the reconstruction. We generated a training set of flat-colored lined sketches with 512 random view samples with the same characteristics as the reference set. Then, we obtained reconstructions from the first 4, 8, 16, 32, 64, 128, 256, and 512 samples, respectively. Each data size was reconstructed with remeshing and without it, as well as in 30+3 iterations and a custom number of iterations. In the second case, the number of iterations was computed considering the number of samples and the batch size to keep the number of parameter updates constant. Equation 22 was used to determine the number of iterations given the number of samples n_s and the desired number of updates $N_u = 960 + 96$, which we computed from the reference configuration.

$$f(n_s) = \frac{4N_u}{n_s} \tag{22}$$

Therefore, we used 960+96, 480+48, 240+24, 120+12, 60+6, 30+3, 15+2, and 7+1 iterations, respectively, in the second case, applying remeshing every 64, 32, 16, 8, 4, 2, 1, and 1 iterations when used. The remeshing period was computed to keep the ratio with the number of iterations constant. We evaluated the obtained reconstructions over the validation reference set.

For brevity, only the reconstructions for Axolotl are shown in Figure 35. Nonetheless, the ones for Vasque follow the same trend. Similarly, Table 6 shows the metrics obtained for Axolotl when using remeshing and a fixed number of iterations for reference. As expected, we observe how the quality increases with the number of samples and iterations. This is sensible as, with more iterations, the results can be further refined, and with a higher number of diverse samples, the ambiguity regarding its shape and appearance is reduced. Good results are obtained with 32 samples and enough iterations, not improving significantly for more than 256 samples. The fact that the improvement in quality reaches a limit is also reasonable, as once enough references are available, new samples do not provide any meaningful novel viewpoints.

Model	Axolotl							
Iterations	30+3							
Samples	4	8	16	32	64	128	256	512
MSE ↓	**0.017**	0.012	0.009	0.008	**0.007**	**0.007**	**0.007**	**0.007**
PSNR ↑	**17.79**	19.38	20.71	21.36	21.64	21.79	21.96	**21.98**
SSIM ↑	**0.904**	0.910	0.917	0.921	0.922	**0.923**	**0.923**	**0.923**
LPIPS ↓	**0.125**	0.121	0.118	0.113	0.112	0.108	0.108	**0.107**
Chamfer ↓	**0.053**	0.067	0.084	0.084	**0.085**	0.083	0.084	0.084
Scaled Chamfer ↓	**0.036**	0.026	0.012	**0.008**	**0.008**	**0.008**	**0.008**	**0.008**

Table 6. Validation metrics of the Axolotl reconstructions from different dataset sizes.

The results were obtained from the Axolotl reference validation set for reconstructions generated with increasing numbers of reference samples.

Model	Axolotl			Vasque		
Displacement	0.2	0.5	0.8	0.2	0.5	0.8
MSE ↓	**0.013**	0.024	**0.032**	**0.039**	0.067	**0.104**
PSNR ↑	**18.85**	16.37	**15.01**	**14.13**	11.77	**9.86**
SSIM ↑	**0.916**	0.915	**0.914**	**0.773**	0.757	**0.751**
LPIPS ↓	**0.125**	**0.131**	0.130	**0.222**	0.259	**0.263**
Chamfer ↓	**0.051**	**0.023**	0.037	**0.085**	**0.138**	**0.085**
Scaled Chamfer ↓	**0.020**	0.032	**0.044**	**0.045**	0.111	**0.150**

Table 7. Validation results of reconstructions with increasing camera inconsistencies.

The results were obtained from the reference validation sets using models reconstructed with increasing levels of camera displacement without remeshing. The best and worst values are depicted in green and red, respectively.

The camera inconsistency was simulated by disturbing camera positions randomly. This was done by generating a random unitary vector to define the direction of the displacement for each camera individually. Then, a given displacement value was used to set the magnitude of this vector before adding it to the camera position. In this way, the resulting cameras were not looking at the center, while our system always expects the look-at point to be at the origin. This effectively introduces a discrepancy between the real view and the view used in the optimization, reproducing camera inconsistencies.

We experimented with displacements of 0.2, 0.5, and 0.8. The datasets generated for each value featured the same statistics as the reference set: 128 training samples of 512 by 512 pixels in a flat-colored lined sketch style. Additionally, the reconstructions were also evaluated using the reference validation set. The results can be seen in Figure 36, while Table 7 presents the metrics after 30+3 iterations without remeshing.

Through these results, we can observe how the reconstructions quickly degrade with the increase of camera discrepancy. This is caused by the fact that color positions and silhouette positions inside the ground truth images are not consistent between shots relative to the optimization cameras due to inconsistency. Therefore, the results are averaged in the view space, meaning that the resulting reconstruction is the shape and color portion commonly seen by all views. In other words, the "consistent" part of the target in relation to the camera. This explains the fast degradation, as small shifts of the camera translate into significant shifts of the target relative to its position in the view.

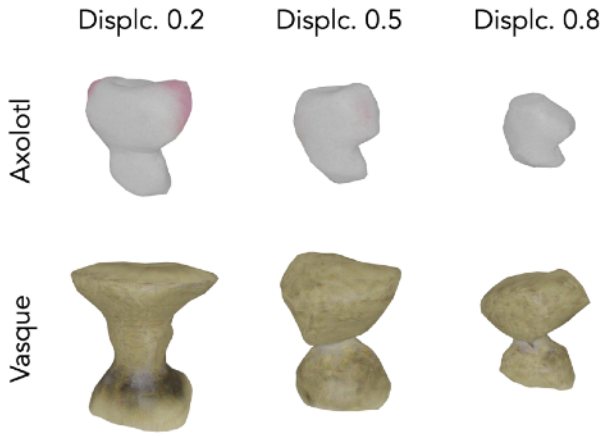

Figure 36. Results without remeshing of increasing levels of camera inconsistency.

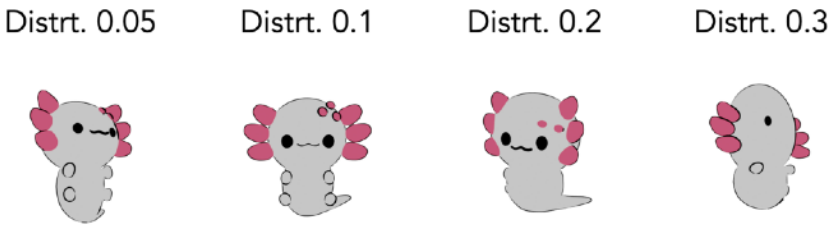

Figure 37. Examples of samples simulating geometry inconsistency.

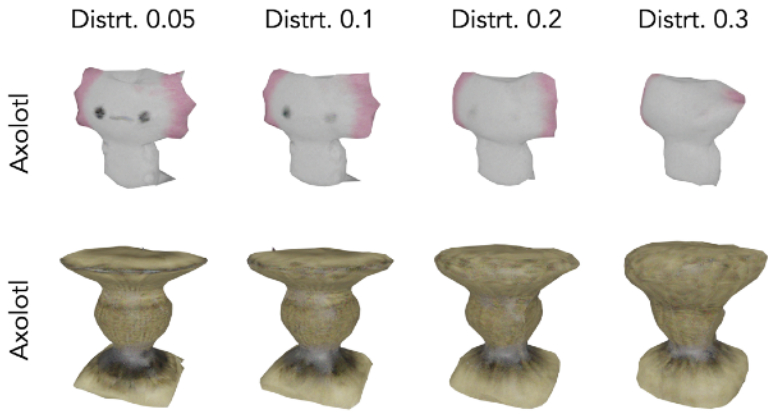

Figure 38. Results without remeshing of increasing levels of geometric inconsistency.

Next, random scaling factors were applied to the meshes of the models before rendering each sample to simulate the geometric inconsistencies. Similarly to the camera inconsistency, this scaling was based on adding a random unitary vector multiplied by the scaling value to the object's scale. Figure 37 shows examples of the samples generated.

Magnitudes of scaling of 0.05, 0.1, 0.2, and 0.3 were evaluated while the datasets generated for each value followed the reference set's statistics, presenting the same number of samples, resolution, and style. The metrics for their reconstructions evaluated over the reference validation set after 30+3 iterations and without remeshing are presented in Table 8, while Figure 38 shows the reconstructions visually.

Model	Axolotl				Vasque			
Distortion	0.05	0.1	0.2	0.3	0.05	0.1	0.2	0.3
MSE ↓	**0.008**	0.009	0.011	**0.013**	**0.024**	0.026	0.030	**0.039**
PSNR ↑	**21.31**	20.57	19.56	**18.91**	**16.29**	15.88	15.22	**14.21**
SSIM ↑	**0.920**	0.918	**0.917**	**0.917**	**0.785**	0.776	0.769	**0.756**
LPIPS ↓	**0.108**	0.114	0.119	**0.121**	**0.191**	0.203	0.213	**0.242**
Chamfer ↓	**0.056**	0.055	0.052	**0.049**	**0.079**	0.080	0.094	**0.133**
Scaled Chamfer ↓	**0.015**	0.015	0.019	**0.022**	**0.025**	0.028	0.038	**0.063**

Table 8. Validation results of reconstructions with increasing geometric inconsistencies.

The results were obtained from the reference validation using reconstructions from samples with increasing levels of geometric distortion without remeshing. The best and worst values are depicted in green and red, respectively.

We can observe how the increase in geometric inconsistency between frames reduces the quality of the result. However, this degradation is lower when compared to the camera inconsistency. This is because, in this case, the degradation results from averaging all the different shapes in the world space. This is consistent with the intended averaging behavior of the silhouette loss under inconsistencies, as presented in Section 4.3.2.

Next, we studied the effects of the image resolution in the reconstruction by generating datasets with the same properties as the initial colored without lines dataset but with different resolutions. This was done to avoid the interference of the line thickness in the samples as for lower resolutions the lines generated obscured the target.

Therefore, we produced sets of samples in the flat-colored style without lines of resolutions of 16 by 16, 32 by 32, 128 by 128, and 1024 by 1024 pixels, comparing them to the initial baseline results from 512 by 512 pixels in the same style. The reconstructions were generated from 128 samples in 30+3 iterations, shown in Figure 39. Additionally, the metrics for Vasque are shown in Table 9, based on the baseline validation set for flat-colored sketches without lines.

As usually happens with image-based methods, the quality of the results decreases with the decrement in resolution, as the amount of information contained in the images is reduced. Despite this tendency, good results are already obtained from a resolution of 128 by 128 pixels. This means that high resolutions are not necessarily required, placing our system in a competitive range as reference sketches are usually of much higher resolution than our required minimum. However, it is also important to note that high resolutions increase the quality mildly, mainly improving color. This is reasonable as we optimize low-resolution meshes and, therefore, the sampling frequency

required to locate a single vertex per pixel and allow a finer optimization is lower. Thus, shape estimation can only take advantage of higher resolutions if a more detailed mesh is used.

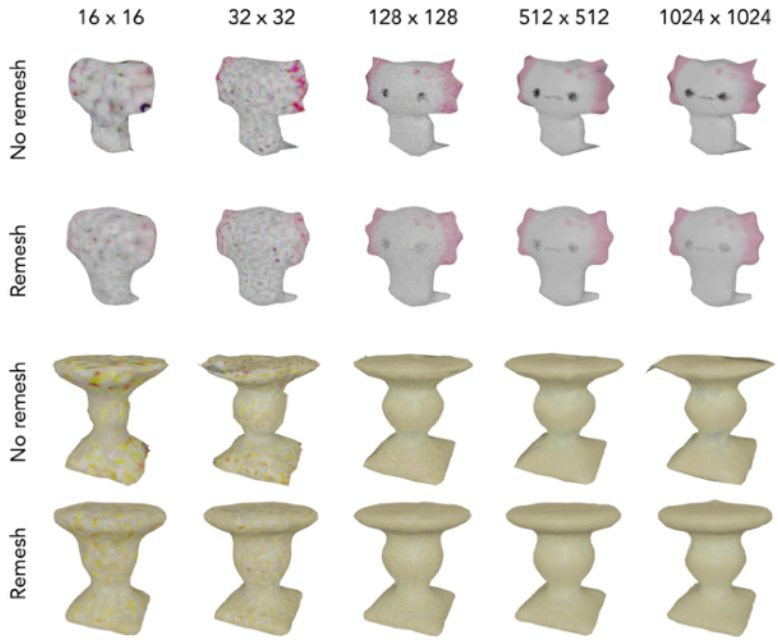

Figure 39. Results from different resolutions.

Model	Vasque									
Remeshing	No					Yes				
Resolution	16	32	128	512	1024	16	32	128	512	1024
MSE ↓	**0.017**	0.015	0.010	**0.009**	0.011	**0.015**	0.013	**0.010**	**0.010**	0.011
PSNR ↑	**17.67**	18.37	20.17	**20.57**	19.73	**18.21**	19.01	19.92	**20.06**	19.75
SSIM ↑	0.830	0.824	**0.819**	0.871	**0.888**	**0.815**	0.819	0.823	0.870	**0.889**
LPIPS ↓	**0.261**	0.246	0.218	0.191	**0.190**	**0.250**	0.244	0.218	**0.190**	**0.190**
Chamfer ↓	0.069	**0.065**	**0.076**	**0.076**	0.070	**0.136**	0.114	0.116	**0.108**	0.109
Scaled Chamfer ↓	**0.039**	0.026	0.024	**0.023**	0.024	**0.058**	0.037	**0.030**	**0.030**	0.033

Table 9. Validation metrics of Vasque reconstructions from different image resolutions.

The results were obtained from the Vasque flat-colored without lines validation set with reconstructions from sketches of increasing squared pixel resolution. The best and worst values are depicted in green and red, respectively.

Finally, the mask precision was studied by generating a set whose samples presented randomly eroded and dilated masks, as in Figure 40. This sample set also followed the reference set statistics, using 30+3 iterations for the reconstruction. Figure 41 visually shows the results.

It can be seen that the effects caused by the noise in the masks are minor in this case. This is because the mask effects get averaged, mainly canceling each other out due to the random presence of bigger and smaller masks than the target. However, we can see their impact on the loss of detail on the sides of Axolotl's head and the loss of roundness in the central section of Vasque.

Figure 40. Examples of references with respectively eroded and dilated masks.

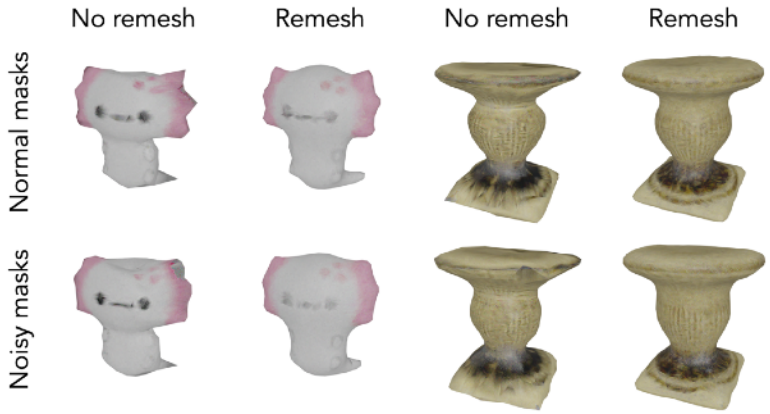

Figure 41. Results from altered masks.

Therefore, after studying different quality factors, we can conclude that the most influencing components are the number of samples used and the camera inconsistencies, followed by the geometric inconsistencies. The former imposes a challenge that is difficult to solve for our approach. Given the optimization-based nature of the system, enough meaningful samples are required for proper optimization. However, this is a limitation of the system, as providing 32 multi-view samples of a given hand-drawn target is unviable in real scenarios.

Meanwhile, the latter factor is related to the need for our rendering-based system for consistent viewpoint definitions. Consequently, the fact that the system is highly dependent and sensitive to errors in the viewpoints is understandable. Still, it imposes a considerable limitation as camera poses for arbitrary drawings are extremely difficult to specify.

5.2.4. Ablation study

To evaluate the design choices of our proposal, we performed an ablation study. We analyzed the contribution to the reconstruction of our proposed resampling, the global illumination, and the defined losses. In these experiments, we used the reference configuration and dataset, taking their results as points of comparison. The reconstructions obtained from the study are presented in Figure 42, while Table 10 complements them with the detailed metrics for spring regularization, smoothness regularization, and global lighting ablations. The metrics of the remaining losses are omitted for brevity, as the results are more clearly portrayed in Figure 42.

For performing the ablation study of the losses, we observed the consequences of removing each one in isolation. Additionally, we checked the effects of not splitting the silhouette and color losses. In our implementation, the shape information is omitted in the color loss by premultiplying both input images with the ground truth mask, hiding the currently optimized silhouette, and obtaining color feedback only from the relevant area. This allows for a further split between the color and

shape losses. Therefore, we studied the unsplit version by removing the shape loss and multiplying each image with its mask for the color loss, introducing the silhouette information together with the color information.

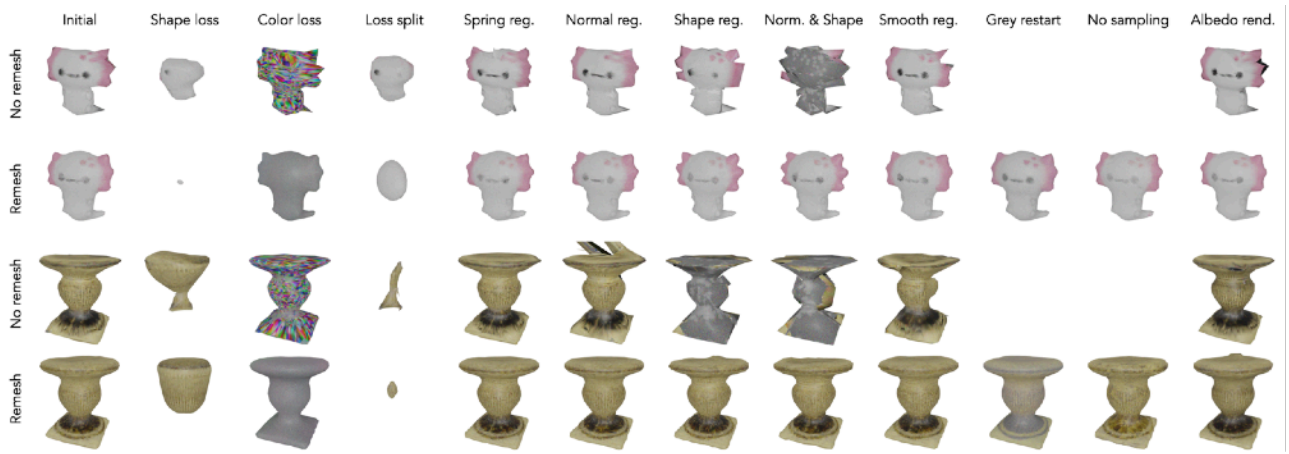

Figure 42. Reconstructions of the reference set under different ablations.

Model	Axolotl						Vasque					
Style	Spring		Smoothness		Lighting		Spring		Smoothness		Lighting	
Remesh.	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
MSE ↓	0.009	0.007	0.007	0.007	0.008	0.007	0.025	0.026	0.030	0.027	0.024	0.026
PSNR ↑	20.64	21.98	21.46	21.86	21.31	21.84	16.16	15.92	15.21	15.74	16.17	15.86
SSIM ↑	0.916	0.922	0.921	0.923	0.928	0.934	0.793	0.789	0.783	0.787	0.796	0.792
LPIPS ↓	0.116	0.107	0.109	0.109	0.108	0.103	0.189	0.195	0.208	0.196	0.184	0.192
Chamfer ↓	0.048	0.083	0.053	0.082	0.055	0.080	0.091	0.113	0.063	0.114	0.077	0.127
Scaled Chamfer ↓	0.016	0.008	0.016	0.008	0.013	0.008	0.024	0.031	0.026	0.032	0.024	0.044

Table 10. Validation results for reconstruction with different ablations.

The results were obtained from the reference validation with the ablated reconstructions without the spring regularization, smoothness regularization, and global lighting. The colors compare the values with the ones in the "Color + lines" columns for each object in Table 5. Green represents an improvement, while red represents a deterioration.

We also compared the results obtained from optimization using albedo rendering with the initial reconstructions using path tracing. This was done to determine if path tracing was genuinely beneficial for our task, complementing the results seen in Section 5.1. Given the entirely white environment, we chose albedo rendering as it is the closest local rendering available without using light sources in *pyredner*. Finally, we analyzed the effects of our proposed sampling method for color recovery by replacing it with the restarting strategy in [20] and by not applying any explicit operation to repair the colors.

The results clearly show how the silhouette loss captures the shape while the color loss captures the colors, being the joint loss not enough to register the shape properly. Therefore, the split loss design is effective. It can also be seen how geometry-related regularizations do not significantly contribute

when using remeshing in this case, as this process already fixes artifacts and produces smooth shaping. However, with less frequent remeshing intervals or more inconsistent samples, we observed that these regularizations might be needed to avoid exaggerated degenerations that are difficult to solve in remeshing.

The normal and shape regularizations help control the mesh and are complementary to each other, explaining why the results do not strongly degenerate when one is still present. However, both are needed as artifacts start appearing when removing any of them. The lack of spring regularization can concentrate more triangles where detail is required but also can cause a higher stretching in them, improving the color in some areas but blurring others. Therefore, the trade-off is not always worthwhile, as Table 10 shows. Lastly, the color smoothness regularization has minimal effects, but they are generally positive, and it induces a better geometry distribution by forcing homogeneity in the triangle's color, as shown by the scaled Chamfer distance metrics. Even though this was an unexpected effect, it is reasonable as limiting the color variation inside the triangle tends to reduce its flexibility for color detail representation, forcing the optimization to place the triangles more smartly.

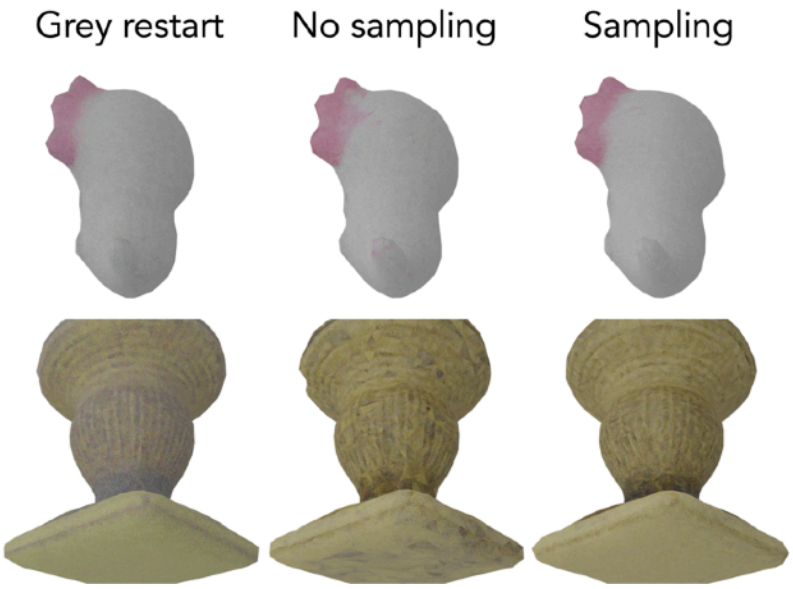

Figure 43. Close-ups of the results obtained with different color repair methods.

The first corresponds to the restart method proposed by Goel et al., the second method does not apply any operation to repair the color, and the last corresponds with our proposal.

Comparing the grey restart [20] with our color sampling proposal, the former washes out colors, being the grey tone still noticeable. Meanwhile, not repairing the colors and leaving the system to refine them again automatically gives a close result to our approach. This is explained when considering that most shuffled triangles will have a similar color to the original ones, reducing the amount of color refinement required in most cases. However, mismatched color patches and higher bleeding still appear, corresponding to the most discrepant instances resulting from shuffling, while our approach significantly reduces these effects. Figure 43 shows a closer look at this comparison.

Finally, with albedo rendering, a higher contrast is obtained for the estimated color improving the image metrics –especially for Vasque– although the results are similar. However, as seen with the remeshed Vasque, it can lead to open surface holes, coinciding with the results seen in Section 5.1. Therefore, we consider it worth using global lighting to identify geometric artifacts better and

reduce visual ambiguity. Nonetheless, albedo rendering can be a good alternative if lower optimization times are required, allowing reconstruction in about 80 minutes compared with about 120 min with path tracing for the reference configuration.

5.2.5. Comparison with inverse rendering techniques

This section compares our proposed system's performance against SFT [20] and NVDiffRec [52]. The reason for this comparison is that our system presents an adaptation of inverse rendering optimization methods for realistic images to the domain of sketches. Therefore, to determine if our efforts were successful, we need to compare how our proposal performs over drawings in relation to standard systems not adapted to the domain.

Model	Axolotl								
Style	B		NC		C6			CM	
System	Ours	NVDR	Ours	NVDR	Ours	NVDR	SFT	Ours	NVDR
MSE ↓	0.006	**0.004**	**0.006**	0.007	0.008	**0.007**	0.033	0.020	**0.017**
PSNR ↑	22.41	**23.70**	**22.61**	21.58	21.08	**21.67**	14.82	17.00	**17.83**
SSIM ↑	0.924	**0.947**	0.925	**0.926**	0.917	**0.929**	0.907	**0.912**	0.896
LPIPS ↓	0.104	**0.086**	0.142	**0.141**	0.107	**0.099**	0.128	0.133	**0.125**
Chamfer ↓	0.068	**0.049**	0.065	**0.032**	0.058	**0.047**	0.048	0.033	**0.025**
Scaled Chamfer ↓	0.005	**0.003**	**0.005**	0.016	**0.006**	0.010	0.064	**0.021**	0.022
Model	Vasque								
Style	B		NC		C6			CM	
System	Ours	NVDR	Ours	NVDR	Ours	NVDR	SFT	Ours	NVDR
MSE ↓	**0.026**	0.033	0.030	**0.026**	**0.037**	0.043	0.095	0.067	**0.041**
PSNR ↑	**15.96**	14.85	15.35	**15.88**	**14.41**	13.81	10.28	11.77	**13.87**
SSIM ↑	0.791	**0.817**	0.793	**0.821**	0.779	**0.783**	0.746	0.757	**0.773**
LPIPS ↓	**0.185**	0.195	0.239	**0.214**	**0.202**	0.214	0.251	0.260	**0.244**
Chamfer ↓	0.073	**0.063**	0.077	**0.065**	0.056	0.086	**0.044**	0.136	**0.072**
Scaled Chamfer ↓	0.024	**0.017**	0.024	**0.018**	**0.024**	0.031	0.086	0.110	**0.023**

Table 11. Validation results for reconstructions with our system, [52], and [20].

The metrics were obtained from the reference validation set with reconstructions from the reference set (B), the lined set without color (NC), a set with six canonical views (C6), and the set with a camera displacement of 0.5 (CM).

It is important to note that, in the present work, we will not compare our approach against other multi-view reconstruction systems for sketches. This is because we are not limiting ourselves to plain-lined sketches. Instead, we also consider color and samples in a potentially higher completeness state, not having exactly the same objective as previous works. This difference in focus made us not directly compete with other multi-view reconstruction methods inside the sketch domain but rather to compete for adapting better in comparison to other inverse rendering approaches.

Furthermore, the comparison should be done against the works reported in Section 2.1.3, which entails some problems. These works depend on generative models, requiring previous training to work. Meanwhile, our approach is fully optimization-based, not needing prior training. Establishing a fair comparison between generative and optimization-based methods is a challenging task, as it is difficult to isolate the effects of the training data in the results of the former. This study would have required the generation of a custom training dataset for these systems that helped accomplish the fairness of the comparison. Even though this is a highly interesting task, it seemed to exceed the scope and time available for our development. Consequently, we decided to focus on the comparison we present now.

The synthetic datasets were used to compare how the systems react to different simulated conditions common in hand-drawn sketches. In particular, we evaluated the reconstructions for the reference set, the lined sketches without color, the set with a camera displacement of 0.5, and a set with only six canonical views located at the sides of a cube centered in the origin.

The configurations were the following. Our system used 30+10 iterations, applying remeshing for Axolotl every two iterations and no remeshing for Vasque. The normal, shape, spring, and smooth regularizations were removed when using remeshing following the observations in Section 5.2.4. For NVDiffRec, we used 5000 iterations, a fixed white environment, a grid of resolution 128, and the remaining default parameters. Lastly, for SFT, we used our initial meshes, a limit of 2048 triangles, and 12 cycles. For the first ten cycles, the iterations for material –diffuse color and roughness– and geometry were limited to five and 150, respectively. From that point on, the limits were set to 75 and 300. Learning rates of 0.01 and $5 \cdot 10^{-4}$ were used for material and geometry, respectively.

Figure 44. Reconstructions with SFT from the sets with six canonical views.

The reconstructions were evaluated using the reference validation set –except for the case with no color where the validation lined set without color was used–, considering only diffuse colors. Table 11 displays the results. Due to time constraints, SFT was only assessed for the case with six canonical views, as its execution times were higher than expected. Nonetheless, the available results, seen in Figure 44, show how our system adapts better to sketches than SFT, as it can be seen

how SFT tends to collapse the optimized geometry, not recovering the shape properly. Therefore, this reflects that our split loss captures better geometry and color than SFT's single MSE loss, while our regularizations help guide the optimization.

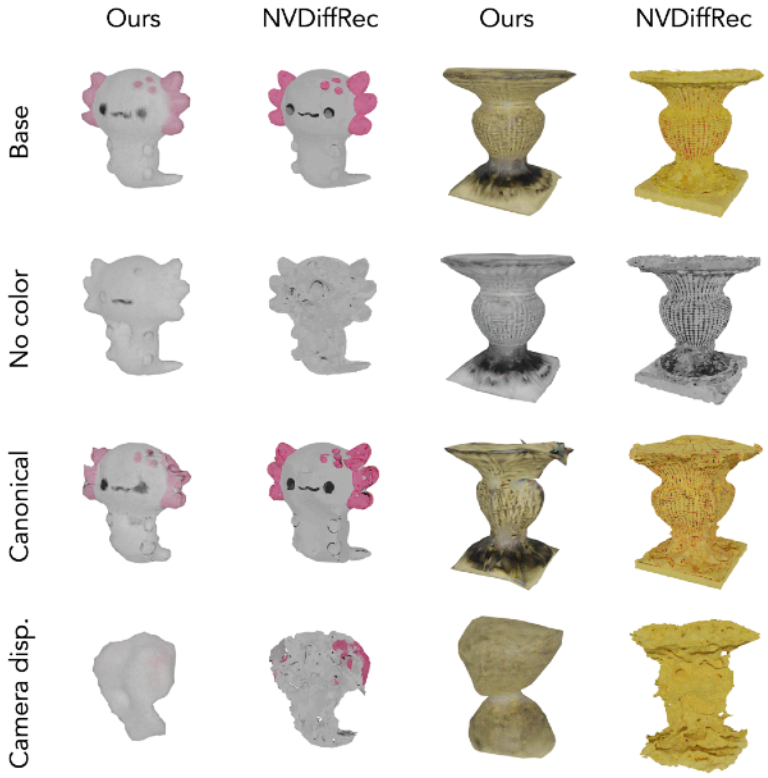

Figure 45. Comparison of the reconstructions obtained by our system and NVDiffRec.

From top to bottom, the reconstructions for different datasets were generated with each system, namely the reference set, the dataset with lined sketches without color, a set with only six canonical views, and the set with a camera displacement of 0.5.

When comparing with NVDiffRec, we observe mixed results, as Figure 45 shows. It is important to note for the comparison that we work with a lower triangle count, presenting our reconstructions 807 triangles on average against 35037 for NVDiffRec. In the base case, we can see that NVDiffRec presents more accurate results thanks to a sharper shape and color estimation. However, in the case of Vasque, we also see how the image metrics are penalized. This is because NVDiffRec estimates specular properties, which interfere with the diffuse color in case of an optimized non-zero specularity. This coincides with the results in Section 5.1 with the dog sketches, where there was ambiguity involving the effects of the material and the environmental light. Therefore, this results further back the use of entirely diffuse materials to avoid this ambiguity.

In the other cases, we observe better performance for our approach under a lack of color and a closer shape estimation under just canonical views. This showcases that, through our method and taking into account the possible inconsistencies in our loss design, we have sacrificed precision in the best possible case while increasing the flexibility of the system. However, it is interesting to observe that the high concentration of lines in Vasque without color guides NVDiffRec towards the proper shape, behaving almost like the presence of color. This reveals the high structural information contained in these lines.

With camera inconsistency, both our proposal and NVDiffRec fail. Even though NVDiffRec presents mostly the best metrics in this case, the reconstructions have higher noise –roughness and

holes– being also notably degraded. This reflects how the high dependence on viewpoints is inherent in the inverse rendering approach.

Finally, thanks to the joint estimation, our proposal presents a temporal cost similar to NVDiffRec under the given configurations, taking two to three hours. Meanwhile, our experiments with SFT on the same hardware have shown times ranging from 10 hours in the canonical case to several days for the remaining cases.

5.2.6. Results from hand-drawn sketches

Finally, to conclude the experimentation over our proposal, we used hand-draw samples to test the system over real scenarios. In particular, we used the manually drawn samples reported in Section 5.2.1. However, before showcasing the results, it should be noted that the 3D models manually designed from the sketches also make compromises regarding the inconsistencies between views to define a coherent shape. Table 12 presents the metrics for each model when comparing their rendered views using our lined and colored sketch pipeline in Blender with the source sketches to quantify this compromise.

Model	Plane	Shrimp	Vase	Abstract
MSE ↓	0.095	0.061	0.058	0.078
PSNR ↑	10.43	12.75	12.56	11.13
SSIM ↑	0.434	0.543	0.579	0.391
LPIPS ↓	0.546	0.554	0.532	0.445

Table 12. Comparison between the sketches and the associated models.

The results were computed as the average among the four views when comparing the rendered human-made models and the sketches.

We reconstructed each object from four canonical samples in 1500+200 iterations, using 20 references for the exportable texture generation, and both without and with remeshing every 100 iterations. Moreover, all regularizations were used in all cases.

In particular, Plane was defined through the top, front, left, and right views; Vase and Shrimp were described through the front, back, top, and side views -right and left respectively-; and Abstract was represented through the top, front, bottom, and right views. It is important to note that the initial mesh for Abstract was estimated using the front, side, and top views as this shape presents relevant hidden features when seen from only two views. Figure 46 shows the reconstruction results.

Once we obtained the reconstructions, we compared them to the human-generated models. We generated 128 renders of the reconstructions and the references from the same random points of view, comparing them to obtain image metrics. Additionally, we also measured the Chamfer distances between reconstructions and models. Table 13 presents these results.

We can see how remeshing generally allows for slightly better reconstructions, as the higher inconsistencies in real sketches can lead the optimization more quickly towards the generation of artifacts. However, when sharper shapes are needed, such as in the Vase's top, not using remeshing allows results better fitting with the silhouettes, with the caveat of some minor degenerations. The results also further show how the reconstructions for Plane and Vase are closer to the references

than the ones for Shrimp and Abstract, which present higher distortion due to the more complex nature of their target shapes. Additionally, Vase showcases how our system can work with shapes of a genus higher than zero.

Finally, we note that areas not seen in the views present random colors, as their color remains unknown to the system. Moreover, when using remeshing, integrated holes, such as in the Plane's wing, can appear. This is caused by degenerations on two near surfaces that lead the remeshing to open a hole increasing the shape's genus. This phenomenon cannot be fixed by only moving the vertices, and it would require redefining the faces around them to remove the hole. Our system is not prepared to apply this kind of face optimization, which is a system limitation.

Model	Plane		Shrimp		Vase		Abstract	
Remeshing	No	Yes	No	Yes	No	Yes	No	Yes
MSE ↓	0.017	**0.016**	0.067	**0.066**	**0.014**	0.015	0.063	**0.062**
PSNR ↑	18.18	**18.48**	12.00	**12.06**	**18.53**	18.31	12.06	**12.14**
SSIM ↑	0.921	**0.923**	0.837	**0.838**	0.927	**0.931**	0.834	**0.841**
LPIPS ↓	**0.085**	0.089	**0.178**	0.183	**0.110**	0.115	**0.190**	0.204
Chamfer ↓	**0.008**	**0.008**	0.037	**0.035**	**0.010**	0.011	0.064	**0.052**
Scaled Chamfer ↓	**0.008**	**0.008**	**0.017**	**0.017**	**0.010**	0.011	0.049	**0.033**

Table 13. Comparison of the reconstructions and the human-made models.

The results were obtained from 128 validation images generated by rendering from random points of view both the reference models and the reconstructions from hand-drawn sketches.

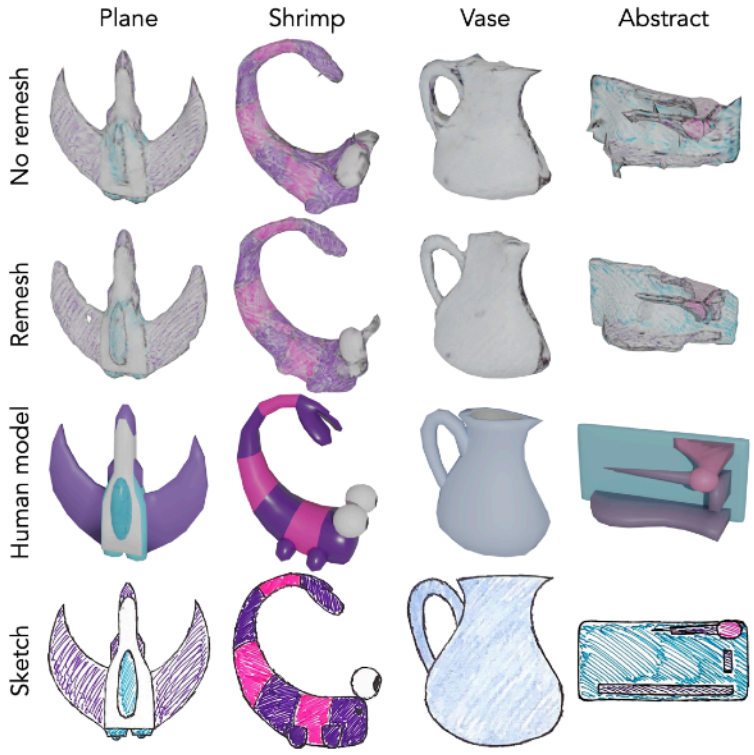

Figure 46. Reconstructions from hand-drawn sketches.

6. Conclusions

Throughout the development of this work, we have studied and practically applied a wide range of state-of-the-art technologies. We have focused mainly on studying the field of reconstruction, both from multi-view sketches and realistic images, analyzing how inverse rendering optimization techniques can be used for drawings. However, this research has allowed us to study and use techniques outside the field to aid our development, such as view pose estimation [67], automatic mask generation [36], and frame interpolation [43]. Additionally, various tools and libraries for mesh manipulation, rendering, image manipulation, and machine learning have been used.

With our first proposal, we have deeply studied one of the most recent technologies in multi-view reconstruction, NVDiffRec [52]. This has allowed us to analyze its principles and architectural designs, obtaining familiarity with the system. From these studies, we have been able to apply NVDiffRec outside the domain initially intended in its creation, using it for reconstruction from illustrations. This had a double objective: studying the capacity for generalization of the system and determining the viability of inverse rendering techniques for sketch reconstruction.

Our qualitative results have shown that NVDiffRec can be used over drawings and allows for promising results, with the strength of recovering detailed silhouettes that maintain the target's essence. However, its use is not ready for real users in our domain, as the meshes obtained from sketches tend to present holes and very abrupt surface topologies. This is related to the kind of internal representation used by NVDiffRec (based on SDFs), the lack of shading in non-colored sketches, and the use of local lighting, which hides the holes and avoids further refinement.

Moreover, because NVDiffRec was designed for realistic images, it presents design choices and components that are not suitable or necessary for drawings. These components are:

- Lighting estimation. In most concept art, objects are depicted without strong shadows, with soft shading, or with no lighting. As we observed in Section 5.1.2 with the dog sketches, using fixed lighting may be more beneficial as it reduces visual ambiguities, such as the origin of the sketch lines.
- Specular texture estimation. In drawings and sketches, specularities are rare and primarily depicted in a non-realistic way. Therefore, the texture details are more desirable to be wholly integrated into the diffuse map, avoiding results such as Figure 28, where the sketch lines are baked into the specular map. Again, removing unnecessary components helps reduce ambiguity and the scope of the problem.
- Perspective camera. As we have shown, an orthographic camera can be more suitable when dealing with non-realistic cases as it can better fit drawings that avoid perspective deformation.
- Local lighting. The rendering pipeline used in NVDiffRec is susceptible to the lack of shading and color, as seen in Sections 5.1.2 and 5.2.5.

Another limiting design choice affecting both NVDiffRec and our second proposal resides in the requirement of camera pose specification. This constraint is caused by the inverse rendering approach, as cameras are essential for the rendering process. However, their sensitivity to mistakes and deviations in the cameras defined compared to the real ones –as hinted through the game character experiments and showcased by the study of quality factors– is highly restrictive when working with illustrations. Unless multi-view sketches are designed following canonical views or turn-around animations, it is highly difficult to specify the viewpoint for an arbitrary drawing.

Despite all this, there are still suitable components in NVDiffRec for drawing reconstruction. The explicit optimization of the mesh, the normal map, and the diffuse map estimations based on rendering are relevant. These are aspects we took into consideration when developing our second proposal.

However, applying our proposed workflow for using NVDiffRec when lacking masks and viewpoints is difficult when working with sketches. COLMAP requires a significant amount of key points for automatic view estimation, which makes its application to actual drawings challenging or even unviable, especially in the absence of any background in the depiction. Similarly, the sometimes abstract nature of sketches can difficult their automatic masking based on segmentation. Therefore, this limits the applicability of our automatic workflow path to highly detailed and completed artworks, which are rare for multi-view setups. The manual workflow will be the most effective solution for typical sketches and flat-colored drawings, increasing user effort.

The results observed with NVDiffRec and its capability to close silhouette matching with the caveat of abnormal surface generation drove us to propose a more tailored solution. Aiming to use global lighting to help avoid the appearance of holes, we built upon the proposal of Goel et al. [20], proposing modifications to the optimization loop, loss functions, and color management after remeshing. These design choices allowed us to be more flexible to the inherent subjective interference in drawings and the uncertainty it causes, showing better behavior than NVDiffRec under limited canonical views and a significant lack of color. Moreover, thanks to an internal mesh representation and global lighting, we can more easily avoid generating open holes in the surface, providing smoother surfaces compared to NVDiffRec. However, under the best conditions with colored sketches, NVDiffRec still allows more detailed and precise reconstructions than our system.

Our results point to the convenience of a split loss to help reduce the mutual ambiguity regarding the effects of color and shape when optimizing from a single loss. This also allows us to capture these properties better and tailor the system to deal with their inconsistencies. Additionally, we have seen how remeshing can solve degenerations but tends to round the shape, which is not always beneficial. Therefore, its use should be considered case by case, being most helpful when working with significant inconsistencies or few samples.

We consider that the results shown by our system are closer to being ready for actual user applications than the ones observed in our first proposal, providing a potential base mesh that artists could use as a reference for testing or building finer models. However, its use is not viable as there are still issues to cover. On the one hand, as Section 5.2.6 has shown, the results over real sketches can still be improved, presenting frequent deformations and not being consistent enough. On the other hand, the large number of samples required to obtain good results is also a limiting factor when considering real use cases. This limitation applies not only to our proposal but to NVDiffRec as well, being a consequence of the optimization nature of the method and the high level of uncertainty that the task involves.

The number of samples and the dependency on camera views are inherent limitations of using inverse rendering optimization techniques over multi-view drawings in real scenarios. However, they are also the most difficult aspects to comply with when working with sketches due to their handmade nature. As seen in Section 5.2.3, other aspects such as the resolution, geometric consistency, or masks' precision have a lower impact on the quality of the reconstruction, being also easier to tackle or avoid. Meanwhile, the set size and the precision of the views have a critical impact, being the most limiting factors of the approach. Therefore, even though these methods show

promising results, overcoming these latter caveats will be a crucial step for effectively applying them to real use cases of the industry.

7. Future work

As presented in Chapter 6, we need to tackle the dependency on camera views and many samples to improve the usability of inverse rendering techniques for reconstruction from sketches. Therefore, ideally, a reconstruction system for drawings should be able to work over a few samples and without explicit viewpoint specification from the user.

With these ideas in mind, we propose leading future research efforts toward designing a preprocessing module capable of transforming the given limited input into an improved input suitable for inverse rendering systems. This would imply the design of a generative module capable of, given the provided input sketches as images, increasing the number of samples by "hallucinating" novel views and consistently regressing their viewpoints.

When considering these ideas, we can find inspiration in the field of novel view synthesis, which tackles generating novel views of an object given its source image. In particular, the works of Zhou et al. [85] and Park et al. [59] present a promising approach. By using novel flow networks, given an image of an object and a desired transformation, they were able to generate a new image of the object from the transformed viewpoint. Moreover, by using the concept of appearance flow, they estimated how to move the pixels in the source image to build the resulting image, forcing consistency of colors and features between them and reducing the degree of hallucination.

By modifying the proposal in [59] to work over the multi-view domain, we could use pixel information from all available views to generate more precise novel views, reducing the need for hallucinating new pixels. In this way, obtaining new views would increase the number of samples available for the reconstruction.

Even though this proposal would require the original viewpoints to work correctly, we envision that a similar model could also be used to regress the image viewpoint. Given that the proposed design can generate a novel view from a reference and a transformation, we theorize that if we could adequately inverse its architecture, obtaining the transformation relating a reference image to a new view would be possible. This concept would allow recovering the relative positions of all the given views by choosing one as the reference, enabling the estimation of the viewpoints. Furthermore, the generative model for novel perspectives could be extended to use the latent representations regarding all the given views and the new view to regress a correction factor for the input transformation, aiming to improve consistency with the generated image.

Of course, this approach should be thoroughly studied to be able to determine its actual viability. Moreover, using a generative module introduces additional considerations. It would require enough precision to avoid introducing further inconsistencies for the reconstruction, and it would raise concerns about generality outside the training domain for the module.

Finally, throughout this work, we have used default rendering pipelines in the optimization. However, the rendering pipeline could be tailored to a non-realistic shading model, trying to reproduce drawing or sketch styles. For this to be viable, a differentiable non-realistic rendering pipeline adapted to our domain should be designed, which can be challenging, as seen in Section 3.1. Alternatively, a deep generative styling module could transform a standard rendered image into a sketch style, avoiding the need for a custom rendering pipeline.

Even though these methods could potentially introduce concerns regarding generality for different illustration styles, they would allow further tailoring to the domain. Therefore, we also consider these alternatives are worth exploring in the future.

8. References

[1] V. Badrinarayanan, A. Handa, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling,” CoRR, vol. abs/1505.07293, 2015, [Online]. Available: http://arxiv.org/abs/1505.07293

[2] S. Bangaru, T.-M. Li, and F. Durand, “Unbiased Warped-Area Sampling for Differentiable Rendering,” ACM Trans. Graph., vol. 39, no. 6, pp. 245:1–245:18, 2020.

[3] D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders,” CoRR, vol. abs/2003.05991, 2020, [Online]. Available: https://arxiv.org/abs/2003.05991

[4] P. Bojanowski, A. Joulin, D. Lopez-Paz, and A. Szlam, “Optimizing the Latent Space of Generative Networks.” arXiv, 2017. doi: 10.48550/ARXIV.1707.05776.

[5] A. Bonnici, A. Akman, G. Calleja, K. Camilleri, P. Fehling, A. Ferreira, F. Hermuth, J. Israel, T. Landwehr, J. Liu, N. Padfield, T. Sezgin, and P. Rosin, “Sketch-based interaction and modeling: where do we stand?,” Artificial Intelligence for Engineering Design, Analysis and Manufacturing, vol. 33, pp. 1–19, Nov. 2019, doi: 10.1017/S0890060419000349.

[6] M. Boss, R. Braun, V. Jampani, J. T. Barron, C. Liu, and H. P. A. Lensch, “NeRD: Neural Reflectance Decomposition from Image Collections.” arXiv, 2020. doi: 10.48550/ARXIV.2012.03918.

[7] C. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, “3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction,” Oct. 2016, pp. 628–644. doi: 10.1007/978-3-319-46484-8_38.

[8] Brent Burley, “Physically Based Shading at Disney,” in SIGGRAPH Courses: Practical Physically Based Shading in Film and Game Production, 2012, pp. 1–27, [Online]. Available: https://disneyanimation.com/publications/physically-based-shading-at-disney/

[9] Colom, Joan and Saito, Hideo, “3D shape reconstruction from non-realistic multiple-view depictions using NVDiffRec,” presented at the Asia-Pacific Workshop on Mixed and Augmented Reality 2022, Yokohama, Japan, 2022. [Online]. Available: https://ceur-ws.org/Vol-3297/paper4.pdf

[10] H. Cui, X. Gao, S. Shen, and Z. Hu, “HSfM: Hybrid Structure-from-Motion,” in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2393–2402. doi: 10.1109/CVPR.2017.257.

[11] B. Curless and M. Levoy, “A Volumetric Method for Building Complex Models from Range Images,” in Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 1996, pp. 303–312. doi: 10.1145/237170.237269.

[12] R. A. Drebin, L. Carpenter, and P. Hanrahan, “Volume Rendering,” in Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 1988, pp. 65–74. doi: 10.1145/54852.378484.

[13] felixyadomi, “Cute Axolotl.” https://sketchfab.com/3d-models/cute-axolotl-e4625a288edf41afab1054a0fa529b3a (accessed Jan. 19, 2023).

[14] Fredo6, “Vasque.” https://3dwarehouse.sketchup.com/model/de673ddf9df03b8278cf1a714198918/Vasque-in-Sketchup (accessed Jan. 19, 2023).

[15] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," Artificial Intelligence Review, vol. 43, no. 1, pp. 55–81, Jan. 2015, doi: 10.1007/s10462-012-9365-8.

[16] C. Gao, Q. Yu, L. Sheng, Y.-Z. Song, and D. Xu, "SketchSampler: Sketch-based 3D Reconstruction via View-dependent Depth Sampling." arXiv, 2022. doi: 10.48550/ARXIV.2208.06880.

[17] J. Gao, W. Chen, T Xiang, C.F. Tsang, A. Jacobson, M. McGuire, and S. Fidler, "Learning Deformable Tetrahedral Meshes for 3D Reconstruction." arXiv, 2020. doi: 10.48550/ARXIV.2011.01437.

[18] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," Atmospheric Environment, vol. 32, no. 14, pp. 2627–2636, 1998, doi: 10.1016/S1352-2310(97)00447-0.

[19] K. Genova and V. Guliashki, "Linear Integer Programming Methods and Approaches–A Survey," Cybernetics and Information Technologies, vol. 11, pp. 3–25, Jan. 2011.

[20] P. Goel, L. Cohen, J. Guesman, V. Thamizharasan, J. Tompkin, and D. Ritchie, "Shape From Tracing: Towards Reconstructing 3D Object Geometry and SVBRDF Material from Images via Differentiable Path Tracing." arXiv, 2020. doi: 10.48550/ARXIV.2012.03939.

[21] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," Advances in Neural Information Processing Systems, vol. 3, pp. 1–9, Jun. 2014, doi: 10.1145/3422622.

[22] Y. Gryaditskaya, F. Hähnlein, C. Liu, A. Sheffer, and A. Bousseau, "Lifting Freehand Concept Sketches into 3D," ACM Trans. Graph., vol. 39, no. 6, Nov. 2020, doi: 10.1145/3414685.3417851.

[23] Y. Gryaditskaya, M. Sypesteyn, J. W. Hoftijzer, S. Pont, F. Durand, and A. Bousseau, "OpenSketch: A Richly-Annotated Dataset of Product Design Sketches," ACM Trans. Graph., vol. 38, no. 6, pp 232:1–232:16, Nov. 2019, doi: 10.1145/3355089.3356533.

[24] F. Hähnlein, Y. Gryaditskaya, A. Sheffer, and A. Bousseau, "Symmetry-Driven 3D Reconstruction from Concept Sketches," in ACM SIGGRAPH 2022 Conference Proceedings, New York, NY, USA, 2022, pp. 19:1–19:8. doi: 10.1145/3528233.3530723.

[25] Z. Han, B. Ma, Y.-S. Liu, and M. Zwicker, "Reconstructing 3D Shapes From Multiple Sketches Using Direct Shape Optimization," IEEE Transactions on Image Processing, vol. 29, pp. 8721–8734, 2020, doi: 10.1109/TIP.2020.3018865.

[26] H. Harbrecht, "Analytical and numerical methods in shape optimization," Mathematical Methods in the Applied Sciences, vol. 31, no. 18, pp. 2095–2114, 2008, doi: https://doi.org/10.1002/mma.1008.

[27] J. Hasselgren, J. Munkberg, J. Lehtinen, M. Aittala, and S. Laine, "Appearance-Driven Automatic 3D Model Simplification." arXiv, 2021. doi: 10.48550/ARXIV.2104.03989.

[28] W. Heidrich, "Computing the Barycentric Coordinates of a Projected Point," J. Graphics Tools, vol. 10, pp. 9–12, Jan. 2005, doi: 10.1080/2151237X.2005.10129200.

[29] A. Hertzmann, "Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines," in ACM SIGGRAPH 99 Course Notes. Course on Non-Photorelistic Rendering, S. Green, Ed., New

York: ACM Press/ACM SIGGRAPH, pp. 7:1–7:14, 1999. [Online]. Available: http://mrl.nyu.edu/publications/npr-course1999/

[30] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, “Mesh Optimization,” in Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 1993, pp. 19–26. doi: 10.1145/166117.166119.

[31] D. S. Immel, M. F. Cohen, and D. P. Greenberg, “A Radiosity Method for Non-Diffuse Environments,” SIGGRAPH Comput. Graph., vol. 20, no. 4, pp. 133–142, Aug. 1986, doi: 10.1145/15886.15901.

[32] J. T. Kajiya, “The Rendering Equation,” in Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 1986, pp. 143–150. doi: 10.1145/15922.15902.

[33] H. Kato, Y. Ushiku, and T. Harada, “Neural 3D Mesh Renderer.” arXiv, 2017. doi: 10.48550/ARXIV.1711.07566.

[34] keith2000, “VisualHullMesh.” Sep. 29, 2021. Accessed: Oct. 30, 2022. [Online]. Available: https://github.com/keith2000/VisualHullMesh

[35] K. Kim, A. Torii, and M. Okutomi, “Multi-view Inverse Rendering Under Arbitrary Illumination and Albedo,” in Computer Vision – ECCV 2016, Cham, 2016, pp. 750–767.

[36] A. Kirillov, Y. Wu, K. He, and R. Girshick, “PointRend: Image Segmentation as Rendering.” arXiv, 2019. doi: 10.48550/ARXIV.1912.08193.

[37] C. Kong and S. Lucey, “Deep Non-Rigid Structure from Motion.” arXiv, 2019. doi: 10.48550/ARXIV.1908.00052.

[38] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Neural Information Processing Systems, vol. 25, Jan. 2012, doi: 10.1145/3065386.

[39] M. Kurt and D. Edwards, “A Survey of BRDF Models for Computer Graphics,” SIGGRAPH Comput. Graph., vol. 43, no. 2, May 2009, doi: 10.1145/1629216.1629222.

[40] S. Laine, J. Hellsten, T. Karras, Y. Seol, J. Lehtinen, and T. Aila, “Modular Primitives for High-Performance Differentiable Rendering.” arXiv, 2020. doi: 10.48550/ARXIV.2011.03277.

[41] C. Li, H. Pan, Y. Liu, A. Sheffer, and W. Wang, “Robust Flow-Guided Neural Prediction for Sketch-Based Freeform Surface Modeling,” ACM Trans. Graph. (SIGGRAPH ASIA), vol. 37, no. 6, pp. 238:1–238:12, 2018, doi: 10.1145/3272127.3275051.

[42] C. Li, H. Lee, D. Zhang, and H. Jiang, “Sketch-based 3D modeling by aligning outlines of an image,” Journal of Computational Design and Engineering, vol. 3, no. 3, pp. 286–294, 2016, doi: 10.1016/j.jcde.2016.04.003.

[43] S. Li, S. Zhao, W. Yu, W. Sun, D. N. Metaxas, C. C. Loy, and Z. Liu, “Deep Animation Video Interpolation in the Wild,” CoRR, vol. abs/2104.02495, 2021, [Online]. Available: https://arxiv.org/abs/2104.02495

[44] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen, “Differentiable Monte Carlo Ray Tracing through Edge Sampling,” ACM Trans. Graph. (Proc. SIGGRAPH Asia), vol. 37, no. 6, pp. 222:1–222:11, 2018.

[45] S. Liu, T. Li, W. Chen, and H. Li, “Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning.” arXiv, 2019. doi: 10.48550/ARXIV.1904.01786.

[46] Z. Lun, M. Gadelha, E. Kalogerakis, S. Maji, and R. Wang, “3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks,” CoRR, vol. abs/1707.06375, 2017, [Online]. Available: http://arxiv.org/abs/1707.06375

[47] P. Mandikal, N. K L, and R. Venkatesh Babu, “3D-PSRNet: Part Segmented 3D Point Cloud Reconstruction From a Single Image,” in Proceedings of the European Conference on Computer Vision (ECCV) Workshops, pp. 662–674, Sep. 2018.

[48] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” arXiv, 2020. doi: 10.48550/ARXIV.2003.08934.

[49] M. Mirbauer, M. Krabec, J. Křivánek, and E. Šikudová, “Survey and Evaluation of Neural 3D Shape Classification Approaches,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 11, pp. 8635–8656, 2022, doi: 10.1109/TPAMI.2021.3102676.

[50] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,” CoRR, vol. abs/1411.1784, 2014, [Online]. Available: http://arxiv.org/abs/1411.1784

[51] T. Müller, A. Evans, C. Schied, and A. Keller, “Instant neural graphics primitives with a multiresolution hash encoding,” ACM Transactions on Graphics, vol. 41, no. 4, pp. 1–15, Jul. 2022, doi: 10.1145/3528223.3530127.

[52] J. Munkberg, J. Hasselgren, T. Shen, J. Gao, W. Chen, A. Evans, T. Müller, and S. Fidler, “Extracting Triangular 3D Models, Materials, and Lighting From Images.” arXiv, 2021. doi: 10.48550/ARXIV.2111.12503.

[53] A. Muntoni and P. Cignoni, “PyMeshLab.” Zenodo, Jan. 2021. doi: 10.5281/zenodo.4438750.

[54] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, “FiberMesh: Designing Freeform Surfaces with 3D Curves,” ACM Trans. Graph., vol. 26, no. 3, pp. 41-es, Jul. 2007, doi: 10.1145/1276377.1276429.

[55] B. Nicolet, A. Jacobson, and W. Jakob, “Large Steps in Inverse Rendering of Geometry,” ACM Trans. Graph., vol. 40, no. 6, pp. 248:1–248:13, Dec. 2021, doi: 10.1145/3478513.3480501.

[56] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, “Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision.” arXiv, 2019. doi: 10.48550/ARXIV.1912.07372.

[57] Y. Ohtake, A. Belyaev, and I. Bogaevski, “Mesh regularization and adaptive smoothing,” Computer-Aided Design, vol. 33, no. 11, pp. 789–800, 2001, doi: https://doi.org/10.1016/S0010-4485(01)00095-1.

[58] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer, “A survey of structure from motion.,” Acta Numerica, vol. 26, pp. 305–364, 2017, doi: 10.1017/S096249291700006X.

[59] E. Park, J. Yang, E. Yumer, D. Ceylan, and A. C. Berg, “Transformation-Grounded Image Generation Network for Novel 3D View Synthesis.” arXiv, 2017. doi: 10.48550/ARXIV.1703.02921.

[60] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, “DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation.” arXiv, 2019. doi: 10.48550/ARXIV.1901.05103.

[61] N. Ravi et al., “Accelerating 3D Deep Learning with PyTorch3D,” CoRR, vol. abs/2007.08501, 2020, [Online]. Available: https://arxiv.org/abs/2007.08501

[62] A. Regnery, “Dog Turnaround Animation,” Behance. https://www.behance.net/gallery/95032661/Dog-Turnaround-Animation (accessed Oct. 19, 2022).

[63] O. Reynolds, Arthur William Brightmore, and William Henry Moorby, The sub-mechanics of the universe, vol. 3. University Press, 1903.

[64] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” CoRR, vol. abs/1505.04597, 2015, [Online]. Available: http://arxiv.org/abs/1505.04597

[65] S. M. Rusinkiewicz, “A New Change of Variables for Efficient BRDF Representation,” in Rendering Techniques ’98, Vienna, 1998, pp. 11–22.

[66] T. Sattler, Q. Zhou, M. Pollefeys, and L. Leal-Taixe, “Understanding the Limitations of CNN-based Absolute Camera Pose Regression.” arXiv, 2019. doi: 10.48550/ARXIV.1903.07504.

[67] J. L. Schönberger and J.-M. Frahm, “Structure-from-Motion Revisited,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4104–4113. doi: 10.1109/CVPR.2016.445.

[68] T. Shen, J. Gao, K. Yin, M.-Y. Liu, and S. Fidler, “Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis.” arXiv, 2021. doi: 10.48550/ARXIV.2111.04276.

[69] M. D. Shivegowda, P. Boonyasopon, S. M. Rangappa, and S. Siengchin, “A Review on Computer-Aided Design and Manufacturing Processes in Design and Architecture,” Archives of Computational Methods in Engineering, vol. 29, no. 6, pp. 3973–3980, Oct. 2022, doi: 10.1007/s11831-022-09723-w.

[70] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, “Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations.” arXiv, 2019. doi: 10.48550/ARXIV.1906.01618.

[71] J. W. H. Tangelder and R. C. Veltkamp, “A survey of content based 3D shape retrieval methods,” Multimedia Tools and Applications, vol. 39, no. 3, pp. 441–471, Sep. 2008, doi: 10.1007/s11042-007-0181-0.

[72] C. Tian, M. Masry, and H. Lipson, “Physical sketching: Reconstruction and analysis of 3D objects from freehand sketches,” Computer-Aided Design, vol. 41, no. 3, pp. 147–158, 2009, doi: https://doi.org/10.1016/j.cad.2009.02.002.

[73] T. S. Trowbridge and K. P. Reitz, “Average irregularity representation of a rough surface for ray reflection,” J. Opt. Soc. Am., vol. 65, no. 5, pp. 531–536, May 1975, doi: 10.1364/JOSA.65.000531.

[74] J. Wang, J. Lin, Q. Yu, R. Liu, Y. Chen, and S. X. Yu, "3D Shape Reconstruction from Free-Hand Sketches," CoRR, vol. abs/2006.09694, 2020, [Online]. Available: https://arxiv.org/abs/2006.09694

[75] L. Wang, C. Qian, J. Wang, and Y. Fang, "Unsupervised Learning of 3D Model Reconstruction from Hand-Drawn Sketches," in Proceedings of the 26th ACM International Conference on Multimedia, New York, NY, USA, 2018, pp. 1820–1828. doi: 10.1145/3240508.3240699.

[76] M. Wang, X.-Q. Lyu, Y.-J. Li, and F.-L. Zhang, "VR content creation and exploration with deep learning: A survey," Computational Visual Media, vol. 6, no. 1, pp. 3–28, Mar. 2020, doi: 10.1007/s41095-020-0162-z.

[77] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction." arXiv, 2021. doi: 10.48550/ARXIV.2106.10689.

[78] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2." 2019. [Online]. Available: https://github.com/facebookresearch/detectron2

[79] C. Xiao, W. Su, J. Liao, Z. Lian, Y.-Z. Song, and H. Fu, "DifferSketching: How Differently Do People Sketch 3D Objects?" arXiv, 2022. doi: 10.48550/ARXIV.2209.08791.

[80] W. Yang, G. Chen, C. Chen, Z. Chen, and K.-Y. K. Wong, "PS-NeRF: Neural Inverse Rendering for Multi-view Photometric Stereo." arXiv, 2022. doi: 10.48550/ARXIV.2207.11406.

[81] C. Yuksel, J. Keyser, and D. H. House, "Mesh Colors," Department of Computer Science, Texas A&M University, 2008.

[82] K. Zhang, F. Luan, Q. Wang, K. Bala, and N. Snavely, "PhySG: Inverse Rendering with Spherical Gaussians for Physics-based Material Editing and Relighting." arXiv, 2021. doi: 10.48550/ARXIV.2104.00674.

[83] S.-H. Zhang, Y.-C. Guo, and Q.-W. Gu, "Sketch2Model: View-Aware 3D Modeling from Single Free-Hand Sketches." arXiv, 2021. doi: 10.48550/ARXIV.2105.06663.

[84] X. Zhang, P. P. Srinivasan, B. Deng, P. Debevec, W. T. Freeman, and J. T. Barron, "NeRFactor: Neural Factorization of Shape and Reflectance Under an Unknown Illumination," ACM Transactions on Graphics, vol. 40, no. 6, pp. 1–18, Dec. 2021, doi: 10.1145/3478513.3480496.

[85] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, "View Synthesis by Appearance Flow." arXiv, 2016. doi: 10.48550/ARXIV.1605.03557.

[86] J. D. Zook, "A simple model for diffuse reflection," Optics Communications, vol. 17, no. 1, pp. 77–82, 1976, doi: https://doi.org/10.1016/0030-4018(76)90183-8.

[87] "GitHub - BachiLi/redner: Differentiable rendering without approximation." https://github.com/BachiLi/redner (accessed Mar. 29, 2023).

[88] "GitHub - colmap/pycolmap: Python bindings for COLMAP." https://github.com/colmap/pycolmap (accessed Apr. 9, 2023).

[89] "GitHub - jpcy/xatlas: Mesh parameterization / UV unwrapping library." https://github.com/jpcy/xatlas (accessed Mar. 29, 2023).

[90] “GitHub - Lightning-AI/torchmetrics: Machine learning metrics for distributed, scalable PyTorch applications.” https://github.com/Lightning-AI/torchmetrics (accessed Apr. 05, 2023).

[91] “GitHub - pmneila/PyMCubes: Marching cubes (and related tools) for Python.” https://github.com/pmneila/PyMCubes (accessed Apr. 05, 2023).

[92] “PyTorch.” https://www.pytorch.org (accessed Mar. 28, 2023).

[93] “TensorFlow.” https://www.tensorflow.org (accessed Mar. 28, 2023).

[94] “Visual Hulls from Uncalibrated Snapshots.” http://www.dip.ee.uct.ac.za/~kforbes/DoubleMirror/DoubleMirror.html (accessed Oct. 30, 2022).