

Tree-Structured Shading Decomposition

Chen Geng*

Hong-Xing Yu*

Sharon Zhang

Maneesh Agrawala

Jiajun Wu

Stanford University

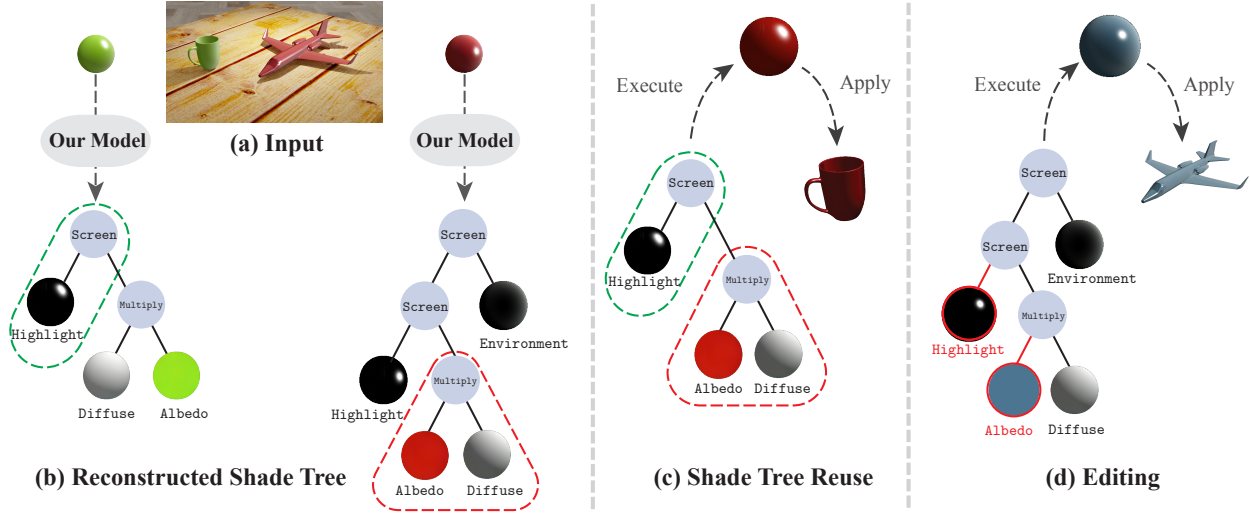


Figure 1. **Decomposing shading into a tree-structured representation.** (a) Our method enables the decomposition of given shading into a (b) *shade tree*. (c) This representation can be reused to generate new *shade trees* and (d) edit the shading of objects.

Abstract

We study inferring a tree-structured representation from a single image for object shading. Prior work typically uses the parametric or measured representation to model shading, which is neither interpretable nor easily editable. We propose using the shade tree representation, which combines basic shading nodes and compositing methods to factorize object surface shading. The shade tree representation enables novice users who are unfamiliar with the physical shading process to edit object shading in an efficient and intuitive manner. A main challenge in inferring the shade tree is that the inference problem involves both the discrete tree structure and the continuous parameters of the tree nodes. We propose a hybrid approach to address this issue. We introduce an auto-regressive inference model to generate a rough estimation of the tree structure and node parameters, and then we fine-tune the inferred shade tree through an optimization algorithm. We show experiments on synthetic images, captured reflectance, real images, and non-realistic

vector drawings, allowing downstream applications such as material editing, vectorized shading, and relighting. Project website: <https://chen-geng.com/inv-shade-trees>.

1. Introduction

Analyzing the shading process in images is fundamental to computer vision and graphics. In particular, the shading process models how the appearances of surfaces are generated from an object’s material properties and lighting conditions. Traditional methods formulate it as the problem of intrinsic decomposition, which expresses the shading as the product of reflectance and albedo [2, 13]. However, this representation is limited in applicability as it assumes a Lambertian surface. Another popular line of works on inverse rendering aims at reconstructing analytical representations [35, 39, 42, 58, 59] or measured representations [30] for materials and lighting. Yet, such physical representations are often difficult to interpret in human perception and not user-friendly for image manipulation tasks.

The choice of shading representation in inverse graphics

*Equal contribution.

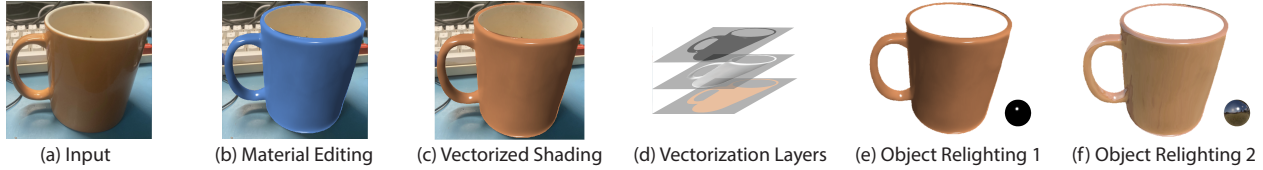


Figure 2. Illustration of downstream applications using the shade tree representation extracted from a single image. For object relighting in (e) and (f), insets show the changed lighting condition.

is important in that it affects what downstream tasks can be accomplished with that representation. The *Shade tree* model is a popular representation for shading in the forward rendering community [7]. One important application of this representation is that it models how vector graphics are shaded [38]. Due to its tree structure, this representation is highly interpretable and easily editable. Thus, it is a worthy and interesting task to recover such representation from visual observations.

In this work, we study recovering the shade tree representation from a single image. We define our shade tree as a binary tree that contains predefined base nodes (like “highlight” and “albedo”) and operations (like “screen mode” and “mix”). Fig. 1 shows examples of our extracted shade trees and subsequently edited materials produced from these extracted trees. In particular, we focus on decomposing the “shading” of objects. The input shading can be considered as spherical reflectance maps or “MatCaps” obtained from existing pipeline [46, 51].

Despite its desirable high interpretability and editability, inferring such a structured representation from a single image has inherent challenges. First, a shade tree contains both continuous parameters for leaf nodes as well as a discrete tree structure, making it difficult to optimize directly. Second, different combinations of base nodes and operations can lead to equivalent structures, introducing additional ambiguities for deterministic inference methods. To infer both discrete structure and continuous parameters, we propose a novel two-stage approach to iteratively decompose an input observation into a shade tree. In the first stage, we use an auto-regressive model to recursively decompose nodes to generate an initial tree structure. Then, we perform sub-structure searching and parameter optimization to fine-tune the tree representation. To deal with the structural ambiguity, we propose a multiple sampling strategy to allow non-deterministic inference that accounts for the multi-modal distribution of plausible shade trees.

Our extensive experiments show the effectiveness of the proposed approach in decomposing the shading of objects using the shade tree representation. Further, we apply our approach to real shadings and non-realistic vector drawings and demonstrate applications on real images. We demonstrate various downstream tasks in Fig. 2.

In summary, our contributions are three-fold. First, we

formulate the problem of inferring *shade trees* from a single image, aiming at understanding the shading of objects with an interpretable representation. Second, we design a novel hybrid approach, integrating an amortized-inference pipeline and an optimization-based solver. Third, we conduct extensive experiments to show the effectiveness of our method and demonstrate potential applications of our method.

2. Related Work

Shade Tree Representation. The history of using shade trees as a rendering representation in computer graphics can be dated back to the 1980s. Cook *et al.* [7] first proposed this representation in 1984, and subsequent use this representation to model the shading of vector graphics [38]. 3D software like Blender [6] uses *node graphs*, a representation similar to *shade trees*.

Few pieces of literature study the problem of inverting such structures. Both Favreau *et al.* [10] and Richardt *et al.* [47] present algorithms to decompose vector graphics into gradient layers, but they do not organize them into tree structures. Lawrence *et al.* [30] study the problem of inverting the parameter of leaf nodes given some fixed shade tree structure. However, our work focuses not only on predicting the parameter of leaf nodes but also on reconstructing the structure of the shade tree.

Shade Trees v.s. Intrinsic Decomposition / BRDFs. Our approach is also related to intrinsic decomposition and inverse rendering.

Intrinsic decomposition methods seek to decompose images into albedo and reflectance in pixel space without further structures [2, 12, 25, 34, 36, 48]. The shading structure recovered in this work is flexible, rather than predefined rules (albedo \times reflectance), differing from common intrinsic decomposition tasks.

Traditional inverse rendering methods aim at recovering material, geometry, and lighting from images [1, 3, 18, 26, 33, 39, 42, 58, 59], using predefined analytical material models such as the Disney BRDF [4]. Compared to the parametric BRDFs, the shade tree focuses on a different level of abstraction. While BRDFs model an element of shading, i.e., *reflectance properties of materials*, it does not model other shading elements such as lighting. Our shade tree models the outcome of shading, i.e., the *appearance*. This involves both

material and lighting for real images, as well as other artistic effects in cartoon shadings. Inverting the shade tree representation features advantages including flexibility in shading, interpretability to common users, and high editability.

Inverse Procedural Graphics. Procedural graphics generates content algorithmically rather than manually. Textures, biological phenomena, and regular structures like buildings and cities are typically generated with procedural models, with a compact set of variables to direct the generation. Inverse procedural graphics seeks to infer parameters or grammar for procedural models describing such structures. This is often done within specific domains, including urban design and layouts [11, 41, 54, 55], L-systems [16, 61], textures [23, 28, 29, 31], forestry [43, 52], CSG (Constructive Solid Geometry) trees [9, 27, 49, 57], and scene representation [32, 37, 40, 56].

Large material datasets [8] coupled with differentiable material graph frameworks [21, 50] have made deep learning methods applicable to procedural material modeling. Given a dataset of training images, Shi *et al.* [50] can select an initial graph structure and optimize its parameters to match a target material appearance. Alternatively, Hu *et al.* [22] directly utilizes the latent space of a generative model to transfer material appearance. In a similar vein, Henzler *et al.* [19] embed images into a latent space before generating BRDF parameters. Our method is different from them in that we simultaneously reconstruct the discrete tree structure and the continuous parameters, allowing better adaptation ability to unseen real images. Generative models have also been applied for creating material representations [17, 60]. More recently, Guerrero *et al.* [15] also shows that transformers are suitable for modeling and generating material graphs, which contain many long-range dependencies. In contrast to generation, we focus on reconstruction from an image.

3. Method

We now introduce our tree decomposition pipeline. First, we introduce the context-free grammar used to represent our shade trees (Sec 3.1). Next, we cover the recursive amortized inference used to produce an initial tree structure (Sec 3.3). Finally, we explain an additional optimization-based fine-tuning step for decomposing any remaining nodes that were not reliably decomposed by the recursive inference (Sec 3.4).

3.1. Grammar Specification

Definition of Shade Trees. A *shade tree* is a tree-structured representation for shading. The leaf nodes of the tree structure are all basic shading nodes that cannot be further decomposed. The interior nodes are formulated using a specified composition method taking child nodes as input. By executing the tree structure in a bottom-up manner, we can get complex shading effects.

Tree	→	Mix(Tree, Tree, Mask)
Tree	→	Multiply(Tree, Tree)
Tree	→	Screen(Tree, Tree)
Tree	→	Albedo(Color=Var)
Tree	→	DiffRef(Lobe=Var, Ambient=Var)
Tree	→	EnvRef
Tree	→	Highlight(Lobe=Var, Sharpness=Var)
EnvRef	→	a environment map
Var	→	free continuous variable
Mask	→	a map with 0 and 1

Table 1. Context-free grammar for the DSL representing *shade tree* structure. More details of the DSL can be found in the supplement.

Definition of Base Nodes. We define four basic shading nodes. Highlight nodes represent a single highlight reflected on the surface. DiffRef nodes represent the diffuse reflective component of the material. Albedo nodes are homogeneous nodes with only one uniform color for shading to represent a basic albedo shading. Finally, EnvRef nodes model the specular shading reflecting the surrounding environment.

Definition of Composition Methods. We define three composing methods to construct parent shading nodes from child nodes. The Multiply operator performs a multiplication of its two child nodes. The Screen operation performs a screen mode composition. The Mix operation takes a mask as input and uses the mask to assign different shading nodes to different regions. For multiply, the shading of parent node p is defined as:

$$p = c_l \cdot c_r, \quad (1)$$

where c_l and c_r denote the left child and the right child, respectively. The screen operation is given by:

$$p = 1 - (1 - c_l) \cdot (1 - c_r). \quad (2)$$

And the mix operation is defined as:

$$p = m \cdot c_l + (1 - m) \cdot c_r, \quad (3)$$

where m denotes a learnable mask.

Context-Free Grammar. The definition of the *shade tree* can be formalized to a domain-specific language (DSL) represented by a context-free grammar [20] G , as shown in Table 1.

3.2. Overview of Algorithm

The proposed algorithm contains two stages. We show an overview in Fig. 3. In the first stage, we aim to recover the initial structure of the *shade tree* using a recursive amortized inference decomposition module (Fig. 3 top). In the

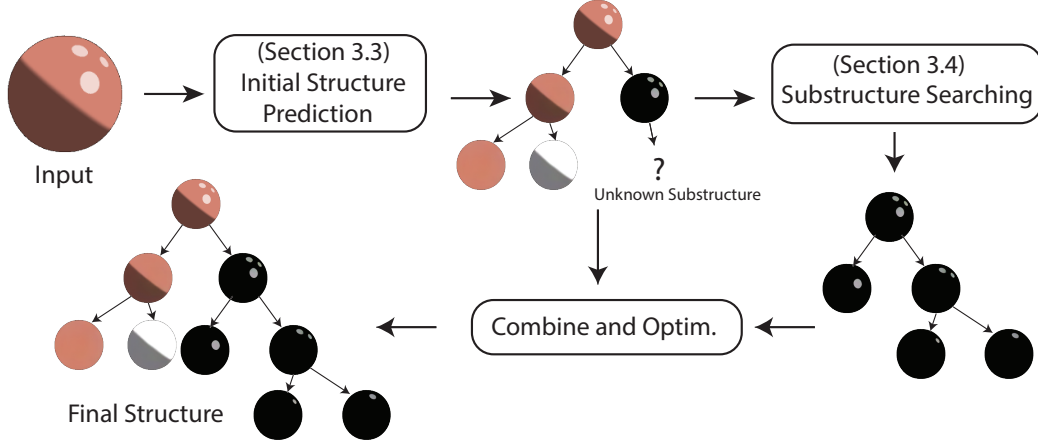


Figure 3. **The proposed framework for reconstructing shade tree representation.** Our method takes in a shading, and then first does an initial structure prediction in a top-down recursive manner. It is allowed that there is still some unknown substructure from this initial guess. Then the roots of those unknown structures are fed into a substructure searching module, where we perform searching over all possible substructures and optimize the leaf parameter to decide whether this structure is appropriate. After all the substructures are decided, we merge them into the initially predicted tree and get the final structure. We perform an overall optimization on this structure to get the final parameter of the leaf nodes.

second stage, we decompose the nodes that are not successfully solved in the first stage and recover the parameters of leaf nodes using an evolution-based optimization algorithm (Fig. 3 bottom).

The motivation for this two-stage design for decomposition is that we wish to take advantage of the distinct behaviors of these two types of algorithms. The first stage is top-down amortized inference and performs the decomposition layer-by-layer. This approach learns prior knowledge from large-scale training data. Thus, the decomposition is fast but occasionally fails in some corner cases due to the lack of enough capacity to generalize, which is seen as a common problem for learning-based methods.

Thus, we further introduce the second stage, which employs a classical program synthesis that enumerates all possible structures and does optimization to find the correct solution. Such an enumeration is slower than learning-based methods, yet it has more capacity to generalize to corner cases. By combining these two approaches, our algorithm is effective and efficient in tackling the task.

3.3. Recursive Amortized Inference

In the first part, we do the decomposition in a top-down manner recursively and then procedurally generate the entire tree. We maintain a pool of nodes and record their type and linkage for each inference procedure. Initially, there is only one node \mathbf{I}_0 in the pool, serving as the root node of the whole tree. At each step, we consider node \mathbf{I} which is neither a leaf nor decomposed. We pass it into our shared single-step component prediction module M and obtain $\{\mathbf{I}_l, \mathbf{I}_r\} = M(\mathbf{I})$, where \mathbf{I}_l and \mathbf{I}_r denote the left and right child nodes, respec-

tively. The design of M will be discussed in this section later. The child nodes \mathbf{I}_l and \mathbf{I}_r are then linked to the parent node \mathbf{I} with new tree edges. All three nodes are then fed into a CNN f which gives

$$\mathbf{p} = f(\mathbf{I}, \mathbf{I}_l, \mathbf{I}_r), \quad (4)$$

where \mathbf{p} is a probability distribution over all compositing operations in our grammar. The operation with the highest probability is selected as the type of the parent node. A separate CNN g that also takes the three nodes as input predicts

$$\Omega = g(\mathbf{I}, \mathbf{I}_l, \mathbf{I}_r), \quad (5)$$

where Ω is the parameter value of the selected operation.

After predicting the operation and corresponding parameters, we then get the reconstructed parent node $\hat{\mathbf{I}}$ by choosing the correct operation from operation set $S = \{\text{mix}, \text{screen}, \text{multiply}\}$ and then get the single-step reconstruction error $\mathcal{L}_{\text{recon}}$:

$$\mathcal{L}_{\text{recon}} = \|\mathbf{I} - \hat{\mathbf{I}}\|_2, \quad (6)$$

$$\text{where } \hat{\mathbf{I}} = S_{\text{argmax}(\mathbf{p})}(\mathbf{I}_l, \mathbf{I}_r, \Omega). \quad (7)$$

To determine whether the predicted child node should be further decomposed, we pass each of $\mathbf{I}_l, \mathbf{I}_r$ into a child component prediction neural network h and get the probability \mathbf{q} of its type. If the child node is a leaf node, then we mark it as solved in the node pool, so it is not further decomposed. The previously described procedure ends whenever no more nodes can be decomposed.

We then describe how the single-step component prediction module M is implemented. The design of M follows two principles:

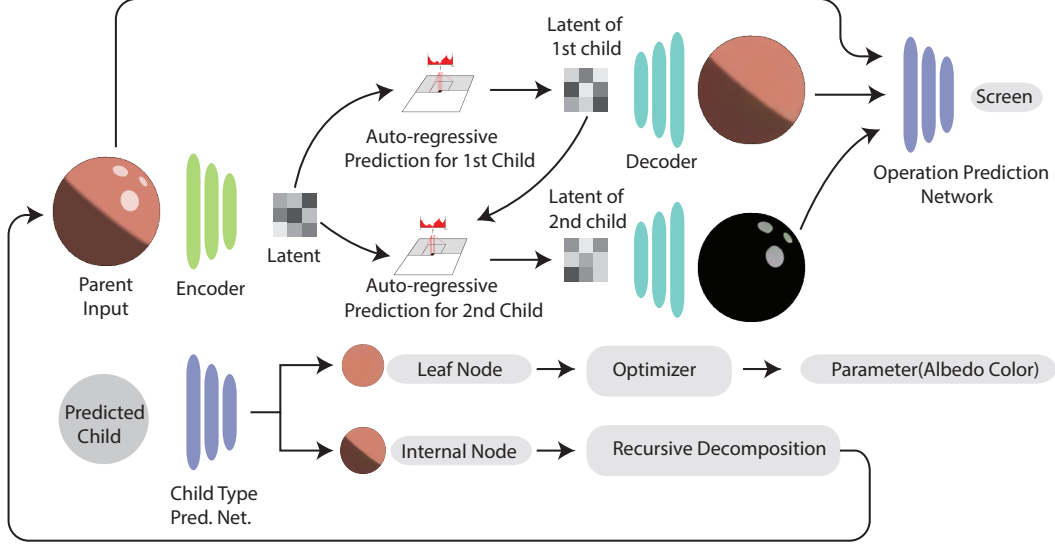


Figure 4. **Illustration of recursive inference module used in the initial structure prediction.** The structure prediction is performed in a top-down recursive manner. In each step, we feed the current parent node into the module, and it is first encoded to a discrete latent code using VQ-VAE. Then the latent code is fed into the first auto-regressive module to predict the latent code for the first child. After that, the latent of the first child and the parent are both fed into the second auto-regressive module to get latent of the second child. The latent codes are decoded into images for children nodes. Afterward, the parent node and the children nodes are fed into the operation prediction network to predict the operation of this step. For each of the predicted children, we use a child-type prediction network to know whether it is a leaf node. If it is not a leaf node, it will be further decomposed. Otherwise, it will be optimized to get its parametric representation.

1. The prediction should not be deterministic, i.e., it should allow different kinds of output for this module. This is because many different structures of trees may describe the same tree.
2. The prediction of the second child should at least depend on the prediction of the first child and the parent node, and the prediction of the first child should depend on the parent node.

Inspired by these two principles, we design a two-step conditional auto-regressive module for prediction.

Auto-regressive Inference. Auto-regressive inference entails a discretized feature space. Thus, we adopt the Vector Quantized Variational Autoencoder (VQ-VAE) [45] as our encoder architecture. The latent feature \mathbf{v} of a shading \mathbf{I} is given by: $\mathbf{v} = E(\mathbf{I})$, where E denotes the VQ-VAE encoder (Fig. 4 left top). Then we further train two conditional Pixel-SNAIL [5] models to auto-regressively generate two child nodes (Fig. 4 middle top). Specifically, for the generation of the first child node, we sample the discrete latent representation \mathbf{v}_l from the distribution $p_l(\mathbf{v})$ represented using the auto-regressive model, conditioned on the latent code of the parent node. Similarly, the latent code \mathbf{v}_r of the second child is sampled from the distributions $p_r(\mathbf{v}, \mathbf{v}_l)$ encoded by the auto-regressive model, conditioned on the previous child and the parent node. Finally, the child images are decoded

using the decoder D to generate the image representation of child nodes.

We build a synthetic dataset to train the previously mentioned modules. Please refer to the supplementary material for the detail of the training.

Multiple Sampling. For each auto-regressive inference, we sample T times to make sure that we make the best decomposition decision in each step. We define a criterion to select from multiple samples. First, we need the reconstruction result that combines two child nodes to be as similar as possible to the parent node, which can be indicated from the $\mathcal{L}_{\text{recon}}$ as described in Eq. 7.

Further, we wish the derived tree to be as compact as possible by avoiding useless decomposition. We define \mathcal{L}_{sim} that represents the similarity between the parent node and the child nodes, which is defined as

$$\mathcal{L}_{\text{sim}} = -\log(\|\mathbf{I} - \mathbf{I}_l\|_2 + \|\mathbf{I} - \mathbf{I}_r\|_2). \quad (8)$$

We also define $\mathcal{L}_{\text{blank}}$ and $\mathcal{L}_{\text{white}}$ to avoid one child being wholly blank or white, which will result in useless decomposition:

$$\mathcal{L}_{\text{blank}} = -\log(\|E(\mathbf{I}_l) - \mathbf{1}\|_2 + \|E(\mathbf{I}_r) - \mathbf{1}\|_2), \quad (9)$$

$$\mathcal{L}_{\text{white}} = -\log(\|E(\mathbf{I}_l)\|_2 + \|E(\mathbf{I}_r)\|_2). \quad (10)$$

The final criterion $\mathcal{L}_{\text{select}}$ is defined as

$$\mathcal{L}_{\text{select}} = \mathcal{L}_{\text{recon}} + \alpha\mathcal{L}_{\text{sim}} + \beta\mathcal{L}_{\text{blank}} + \gamma\mathcal{L}_{\text{white}}, \quad (11)$$

	Realistic			Toon			DRM (real-captured)		
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
CNN	26.50	0.967	0.066	27.18	0.959	0.052	14.41	0.857	0.164
LSTM	17.80	0.909	0.154	24.86	0.964	0.067	19.67	0.882	0.205
Transformer	18.82	0.930	0.153	26.73	0.971	0.042	17.89	0.876	0.186
Ours	30.89	0.974	0.052	30.08	0.972	0.032	25.47	0.927	0.150

Table 2. **Quantitative comparison.** Our method greatly surpasses other baselines in all three datasets, benefiting from our design to accurately predict structure and parameters. “Realistic” and “Toon” are two synthetic datasets and “DRM” is a real-captured dataset.

where α, β, γ are hyper-parameters.

Early-stop Strategy. The amortized inference module may not successfully decompose every node, which is why we designed the second stage to further decompose those nodes and finetune the whole tree structure. We send a node to the second stage if $\min_T(\mathcal{L}_{\text{select}}) < \tau$, where τ is a threshold.

3.4. Optimization-based Finetuning

For the nodes that cannot be decomposed in the first stage, we search over all possible sub-structures of these nodes and use an optimizer BasinCMA [24] to find the optimal leaf parameters. The optimization target can be defined as

$$\min_{\lambda} \|R_s(\lambda) - \mathbf{I}\|_2 + \|F_{\text{VGG}}(R_s(\lambda)) - F_{\text{VGG}}(\mathbf{I})\|_2, \quad (12)$$

where λ denotes all trainable parameters, R_s represents the renderer under the searched structure s and F_{VGG} represents a pretrained VGG network.

The search over all possible substructures is performed in the order of depth. If the target loss is already smaller than a predefined threshold ϕ , we stop searching and assume it to be the final substructure.

Obtaining parametric representation for leaf nodes. After finalizing the shade tree’s structure, we optimize each leaf node using the same target as described in Eq. 12 to get the parametric representation for each leaf node.

Optimizing on the whole tree. Finally, we perform optimization on the parameter of all leaf nodes using the following target:

$$\min_{\mu} \|R_S(\mu) - \mathbf{I}_0\|_2 + \|F_{\text{VGG}}(R_S(\mu)) - F_{\text{VGG}}(\mathbf{I}_0)\|_2, \quad (13)$$

where S stands for the finalized structure of the whole tree and $\mu = [\lambda_0, \lambda_1, \dots, \lambda_N]$.

4. Experiments

In this section, we first show our results on shade tree decomposition on a diverse set of example including realistic synthetic images, cartoon-style images, and real images. Then we introduce a visual shading editing analogy experiment which is designed to quantitatively evaluate the decomposition of reconstructed structures. Finally we analyze our method by ablation study results.

4.1. Results on Decomposition

To evaluate the effectiveness of the proposed method, we conduct a quantitative evaluation of our methods and other baselines on several datasets.

Datasets. We evaluate our method on both synthetic and real-captured datasets.

For the synthetic dataset, we generate two styles of datasets, “Realistic” and “Toon”, to show the robustness and broad applicability of the proposed method. For the “Realistic” dataset, all the base nodes are represented in a photo-realistic way, imitating how the shading in real life behaves. For the “Toon” dataset, we take inspiration from non-photorealistic shading [14] and generate many cartoon-style shading nodes. After generating all base nodes and operation nodes, they are split into two sets, one for training sets and the other for the generation of test sets. Afterward, we apply a recursive algorithm to generate the training and test sets using the specified context-free grammar. The details of the dataset can be found in the supplementary material.

Besides the synthetic datasets, we use the real-captured dataset “DRM” collected by Rematas et al. [46] to evaluate the real-world generalizability of the proposed method.

Baselines. No previous work has tackled a task setting similar to ours. Therefore, we drew inspiration from previous research on grammar decomposition and adapted three competitive baseline frameworks that are widely used in the neural program synthesis and structure induction community for our purpose.

Our **CNN** baseline, which utilizes a similar architecture to that in Rim-net [44], employs an encoder-decoder structure to perform single-step decomposition recursively, similar to the first stage of our approach. We also introduced an **LSTM** baseline, similar to Shape Programs [53], which first uses an encoder to get the latent representation of images and then uses LSTM to predict a sequence of tokens that are subsequently compiled to the shade tree structure. Similarly, our **Transformer** baseline also predicts the sequence of tokens but adapts a GPT architecture, following Matformer [15]. Please note that although the baselines share a similar backbone design with previous literature, they differ due to the different problem settings. The supplementary material

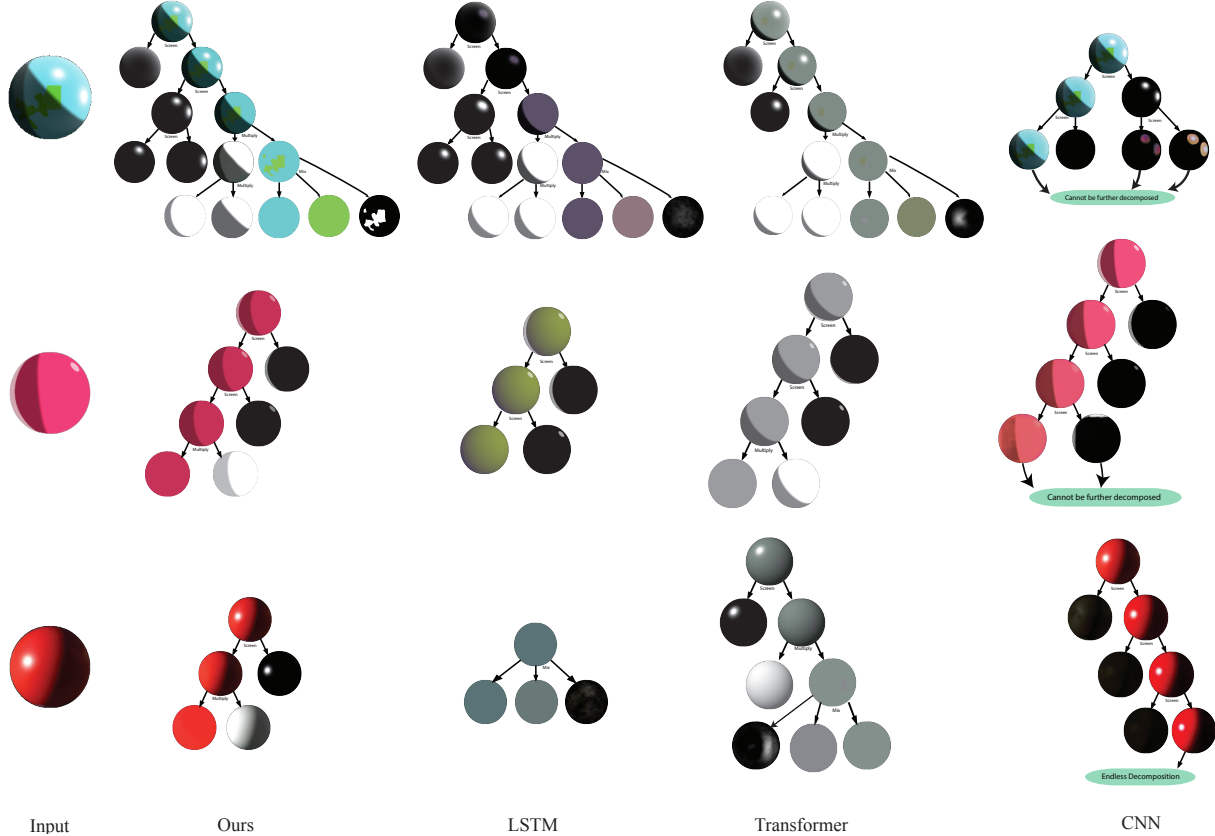


Figure 5. **Qualitative comparison of different methods on reconstruction.** [Top row] shows a sample from *Realistic* dataset, [Mid row] shows a sample from *Toon* dataset, and [Bottom row] shows a real test sample from the *DRM* dataset [46].

contains further details and implementation of the baselines.

Results. We show the results of this experiment in Fig. 5 and Table 2. Our method has the best-reconstructed tree structure among all three methods. The LSTM baseline can predict similar structures to ours; however, it performs poorly in predicting the parameter of leaf nodes, resulting in bad reconstruction results. The CNN baseline predicts in a top-down manner; however, it suffers from ambiguity in the grammar. Thus, it cannot learn a correct mapping between layers, resulting in nodes that cannot be further decomposed or endlessly decomposed in a trivial way.

Our method can also be applied to the real-world dataset “DRM” with satisfactory performance, which can be witnessed from the third column of Table 2 and the third row of Fig. 5. The result shows the generalizability and the real-world applicability of the proposed approach.

We also perform decomposition using our model on some in-the-wild internet photos, shown in Fig. 6.

4.2. Visual Shading Editing Analogy

To allow quantitatively evaluating the reconstructed tree structure, we design a task called “Visual Shading Editing

	PSNR↑	SSIM↑	LPIPS↓
CNN	4.79	0.143	0.608
LSTM	4.47	0.113	0.547
Transformer	5.02	0.186	0.547
Ours	32.17	0.913	0.078

Table 3. **Quantitative comparison of different methods on the task visual shading editing analogy.** Our method performs the best among all three methods by understanding the tree structure well. The other two methods cannot deal with this task because of their poor decomposition.

Analogy” which reflects how well the decomposition is. As illustrated in Fig. 7, given an input pair of shading, the algorithm should give a hybrid shade tree composed of different subtrees from different nodes, according to the rule shown in the example shading ball pair.

We generate a dataset containing different types of shading editing to evaluate the performance of different methods on this task. We adopt the same baseline setting in Section 4.1, introducing the CNN, LSTM, and Transformer baselines. Then we use such methods to decompose given pairs

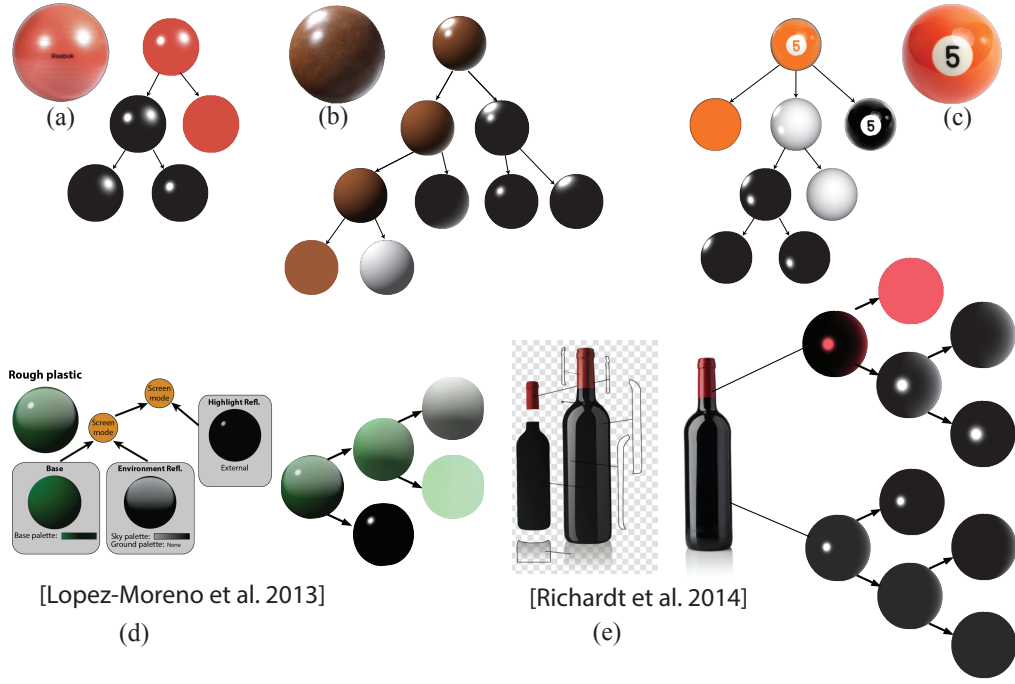


Figure 6. **Decomposition of in-the-wild real images using our method.** Our method can not only work on synthetic data but can also be widely used in the decomposition of in-the-wild shading. The shadings in (a, b, c) are collected from the Internet. In (d), we show that our method can do decomposition of the shadings from Lopez-Moreno et al. [38]. In (e), we compare our method with Richardt et al. [47]. We first extract shading from the vector drawing, and then we use our method to do decomposition to the shading sphere.

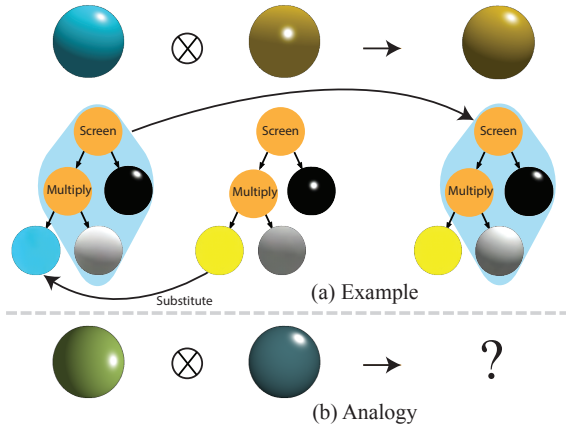


Figure 7. **Illustration of “Visual Shading Editing Analogy” that allows quantitative evaluation.** Given an example of shading editing, we wish the same operation could be applied to novel test pairs. For instance, in this case, the example shows us that the edited shading is formed by replacing the albedo node with the albedo node from the second shading ball. Thus, this operation should also be applied to the pair in the bottom row, resulting in a shading ball with a blue base color and an upward highlight.

to get their tree-structured representation. The details of the algorithm for making such a visual analogy are described in the supplementary material.

Table 3 shows the results. Our method surpasses other methods greatly due to a better understanding of the semantic meaning of tree structure.

4.3. Ablation Studies

To verify the influence of the special design in the proposed pipeline, we do ablation studies on the following three components: multiple sampling, second-stage optimization, and overall optimization, because they are typically non-trivial in previous literature.

Influence of Multiple Sampling From Fig. 8, it can be observed that multiple sampling improves the result during the 1st stage inference because it can produce several solutions and use the metrics to choose the best one. Without doing this, our model may directly predict a “reasonable” one, but not the “best” one.

Influence of Optimization The second stage of optimization help us to decompose those nodes that are hard to deal with using only the amortized-inference module. By removing such a component, our method cannot decompose the bottom node shown in Fig. 8 and thus gives worse results than the

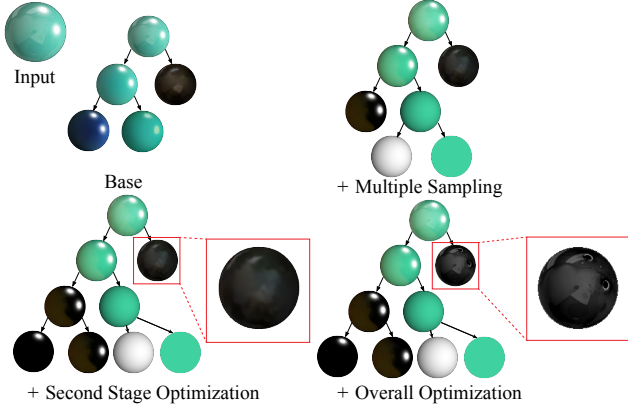


Figure 8. **Ablation of components.** “Base” denotes our method with only 1 sample during inference, “+Multiple Sampling” denotes only using the 1st stage with multiple sampling. “+Second Stage” denotes using both the first stage and the second stage of the proposed method but does not perform the overall optimization. “+Overall Optimization” denotes the full proposed method.

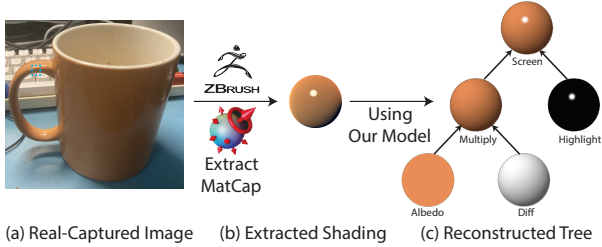


Figure 9. **Results on a real image of a non-sphere object.** Although the main focus of this work is decomposing the reflectance spheres, it can be applied to non-sphere geometry as well if we use existing tools to first extract sphere reflectance from images first.

full method.

Influence of Overall Optimization By introducing the overall optimization at the end of the second stage, our method can further finetune the structure, like giving a better environment reflection.

4.4. Application on Real-world Images

Our work focuses on the decomposition of MatCaps or Reflectance Maps [51]. However, the work can be applied to real-world images by using existing tools to first extract the sphere reflectance. In Fig. 9, we show an example of using the proposed method together with an existing tool ZBrush to decompose the shading of a real-world capture.

5. Conclusion

We have presented a novel method that can effectively and efficiently decompose shading into a tree-structured representation, which enables understanding and editing of

the shading in an interpretable way. The first stage of the proposed method uses a pretrained auto-regressive model to predict the structure and parameters of the tree structure. The second stage of the pipeline leverages the parametric representation of each base node and structure searching to find the optimal structure for all nodes that cannot be effectively decomposed in the first stage. The combination of two stages leads to our state-of-the-art performance on several datasets compared to the baselines.

Acknowledgments. This work was in part supported by Ford, NSF RI #2211258, AFOSR YIP FA9550-23-1-0127, the Toyota Research Institute (TRI), the Stanford Institute for Human-Centered AI (HAI), Amazon, and the Brown Institute for Media Innovation.

References

- [1] Jonathan T Barron and Jitendra Malik. Shape, illumination, and reflectance from shading. *IEEE transactions on pattern analysis and machine intelligence*, 37(8):1670–1687, 2014. 2
- [2] Sean Bell, Kavita Bala, and Noah Snavely. Intrinsic images in the wild. *ACM Transactions on Graphics (TOG)*, 33(4):1–12, 2014. 1, 2
- [3] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. Nerd: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12684–12694, 2021. 2
- [4] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *Acm Siggraph*, volume 2012, pages 1–7. vol. 2012, 2012. 2
- [5] Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. Pixelsnail: An improved autoregressive generative model. In *International Conference on Machine Learning*, pages 864–872. PMLR, 2018. 5
- [6] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 2
- [7] Robert L Cook. Shade trees. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 223–231, 1984. 2
- [8] Valentin Deschaintre, Miika Aittala, Frédo Durand, George Drettakis, and Adrien Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 37(128):15, aug 2018. 3
- [9] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Trans. Graph.*, 37(6), dec 2018. 3
- [10] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Photo2clipart: Image abstraction and vectorization using layered linear gradients. *ACM Trans. Graph.*, 36(6), nov 2017. 2

- [11] Mathieu Gaillard, Vojtěch Krs, Giorgio Gori, Radomír Měch, and Bedrich Benes. Automatic differentiable procedural modeling. *Computer Graphics Forum*, 41(2):289–307, 2022. 3
- [12] Elena Garces, Adolfo Munoz, Jorge Lopez-Moreno, and Diego Gutierrez. Intrinsic images by clustering. In *Computer graphics forum*, volume 31, pages 1415–1424. Wiley Online Library, 2012. 2
- [13] Elena Garces, Carlos Rodriguez-Pardo, Dan Casas, and Jorge Lopez-Moreno. A survey on intrinsic images: Delving deep into lambert and beyond. *International Journal of Computer Vision*, 130(3):836–868, 2022. 1
- [14] Bruce Gooch and Amy Gooch. *Non-photorealistic rendering*. AK Peters/CRC Press, 2001. 6
- [15] Paul Guerrero, Milos Hasan, Kalyan Sunkavalli, Radomir Mech, Tamy Boubekeur, and Niloy Mitra. Matformer: A generative model for procedural materials. *ACM Trans. Graph.*, 41(4), 2022. 3, 6
- [16] Jianwei Guo, Haiyong Jiang, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, Dani Lischinski, and Hui Huang. Inverse procedural modeling of branching structures by inferring l-systems. *ACM Trans. Graph.*, 39(5), jun 2020. 3
- [17] Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. Materialgan: Reflectance capture using a generative svbrdf model. *ACM Trans. Graph.*, 39(6), nov 2020. 3
- [18] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, light & material decomposition from images using monte carlo rendering and denoising. *arXiv preprint arXiv:2206.03380*, 2022. 2
- [19] Philipp Henzler, Valentin Deschaintre, Niloy J Mitra, and Tobias Ritschel. Generative modelling of brdf textures from flash images. *ACM Trans Graph (Proc. SIGGRAPH Asia)*, 40(6), 2021. 3
- [20] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001. 3
- [21] Yiwei Hu, Paul Guerrero, Milos Hasan, Holly Rushmeier, and Valentin Deschaintre. Node graph optimization using differentiable proxies. In *ACM SIGGRAPH 2022 Conference Proceedings*, SIGGRAPH ’22, New York, NY, USA, 2022. Association for Computing Machinery. 3
- [22] Yiwei Hu, Miloš Hašan, Paul Guerrero, Holly Rushmeier, and Valentin Deschaintre. Controlling material appearance by examples. *Computer Graphics Forum*, 41(4):117–128, 2022. 3
- [23] Yiwei Hu, Chengan He, Valentin Deschaintre, Julie Dorsey, and Holly Rushmeier. An inverse procedural modeling pipeline for svbrdf maps. *ACM Trans. Graph.*, 41(2), jan 2022. 3
- [24] Minyoung Huh, Richard Zhang, Jun-Yan Zhu, Sylvain Paris, and Aaron Hertzmann. Transforming and projecting images into class-conditional generative networks. In *European Conference on Computer Vision*, pages 17–34. Springer, 2020. 6
- [25] Michael Janner, Jiajun Wu, Tejas D Kulkarni, Ilker Yildirim, and Josh Tenenbaum. Self-supervised intrinsic image decomposition. *Advances in neural information processing systems*, 30, 2017. 2
- [26] Haian Jin, Isabella Liu, Peijia Xu, Xiaoshuai Zhang, Songfang Han, Sai Bi, Xiaowei Zhou, Zexiang Xu, and Hao Su. Tensor: Tensorial inverse rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2
- [27] R. Kenny Jones, Homer Walke, and Daniel Ritchie. Plad: Learning to infer shape programs with pseudo-labels and approximate distributions. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3
- [28] Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. Procedural noise using sparse gabor convolution. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH ’09, New York, NY, USA, 2009. Association for Computing Machinery. 3
- [29] Ares Lagae, Peter Vangorp, Toon Lenaerts, and Philip Dutré. Procedural isotropic stochastic textures by example. *Computers & Graphics*, 34(4):312–321, 2010. 3
- [30] Jason Lawrence, Aner Ben-Artzi, Christopher DeCoro, Wojciech Matusik, Hanspeter Pfister, Ravi Ramamoorthi, and Szymon Rusinkiewicz. Inverse shade trees for non-parametric material representation and editing. *ACM Transactions on Graphics (TOG)*, 25(3):735–745, 2006. 1, 2
- [31] Laurent Lefebvre and Pierre Poulin. Analysis and synthesis of structural textures. In *Graphics Interface*, volume 2000, pages 77–86, 2000. 3
- [32] Yikai Li, Jiayuan Mao, Xiuming Zhang, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Perspective Plane Program Induction from a Single Image. In *Conference on Computer Vision and Pattern Recognition*, 2020. 3
- [33] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and svbrdf from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2475–2484, 2020. 2
- [34] Zhengqi Li and Noah Snavely. Learning intrinsic image decomposition from watching the world. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2018. 2
- [35] Zhengqin Li, Zexiang Xu, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Learning to reconstruct shape and spatially-varying reflectance from a single image. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 1
- [36] Yunfei Liu, Yu Li, Shaodi You, and Feng Lu. Unsupervised learning for intrinsic image decomposition from a single image. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2020. 2
- [37] Yunchao Liu, Jiajun Wu, Zheng Wu, Daniel Ritchie, William T. Freeman, and Joshua B. Tenenbaum. Learning to describe scenes with programs. In *International Conference on Learning Representations*, 2019. 3
- [38] Jorge Lopez-Moreno, Popov Stefan, Adrien Bousseau, Maneesh Agrawala, and George Drettakis. Depicting stylized materials with vector shade trees. *ACM Transactions on Graphics*, 32(4), 2013. 2, 8
- [39] Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. Unified shape and svbrdf recovery using differentiable monte

- carlo rendering. In *Computer Graphics Forum*, volume 40, pages 101–113. Wiley Online Library, 2021. 1, 2
- [40] Jiayuan Mao, Xiuming Zhang, Yikai Li, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Program-Guided Image Manipulators. In *International Conference on Computer Vision*, 2019. 3
- [41] Andelo Martinovic and Luc Van Gool. Bayesian grammar learning for inverse procedural modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013. 3
- [42] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting triangular 3d models, materials, and lighting from images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8280–8290, 2022. 1, 2
- [43] Till Niese, Sören Pirk, Matthias Albrecht, Bedrich Benes, and Oliver Deussen. Procedural urban forestry. *ACM Trans. Graph.*, 41(2), mar 2022. 3
- [44] Chengjie Niu, Manyi Li, Kai Xu, and Hao Zhang. Rim-net: Recursive implicit fields for unsupervised learning of hierarchical shape structures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11779–11788, 2022. 6
- [45] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019. 5
- [46] Konstantinos Rematas, Tobias Ritschel, Mario Fritz, Efstratios Gavves, and Tinne Tuytelaars. Deep reflectance maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4508–4516, 2016. 2, 6, 7
- [47] C. Richardt, J. Lopez-Moreno, A. Bousseau, M. Agrawala, and G. Drettakis. Vectorising bitmaps into semi-transparent gradient layers. In *Proceedings of the 25th Eurographics Symposium on Rendering*, EGSR ’14, page 11–19, Goslar, DEU, 2014. Eurographics Association. 2, 8
- [48] Carsten Rother, Martin Kiefel, Lumin Zhang, Bernhard Schölkopf, and Peter Gehler. Recovering intrinsic images with a global sparsity prior on reflectance. *Advances in neural information processing systems*, 24, 2011. 2
- [49] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 3
- [50] Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. Match: Differentiable material graphs for procedural material capture. *ACM Trans. Graph.*, 39(6):1–15, Dec. 2020. 3
- [51] Peter-Pike J Sloan et al. The lit sphere: A model for capturing npr shading from art. *Graphics Interface*, 2001. 2, 9
- [52] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Benes. Inverse procedural modelling of trees. *Computer Graphics Forum*, 33(6):118–131, 2014. 3
- [53] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. *arXiv preprint arXiv:1901.02875*, 2019. 6
- [54] Carlos A. Vanegas, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Paul Waddell. Inverse design of urban procedural models. *ACM Trans. Graph.*, 31(6), nov 2012. 3
- [55] Fuzhang Wu, Dong-Ming Yan, Weiming Dong, Xiaopeng Zhang, and Peter Wonka. Inverse procedural modeling of facade layouts. *CoRR*, abs/1308.0419, 2013. 3
- [56] Jiajun Wu, Joshua B Tenenbaum, and Pushmeet Kohli. Neural scene de-rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3
- [57] Q. Wu, K. Xu, and J. Wang. Constructing 3d csg models from 3d raw point clouds. *Computer Graphics Forum*, 37(5):221–232, 2018. 3
- [58] Kai Zhang, Fujun Luan, Zhengqi Li, and Noah Snavely. Iron: Inverse rendering by optimizing neural sdfs and materials from photometric images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5565–5574, 2022. 1, 2
- [59] Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. Physg: Inverse rendering with spherical gaussians for physics-based material editing and relighting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5453–5462, 2021. 1, 2
- [60] Xilong Zhou, Miloš Hašan, Valentin Deschaintre, Paul Guerrero, Kalyan Sunkavalli, and Nima Kalantari. Tilegen: Tileable, controllable material generation and capture, 2022. 3
- [61] O. Štřava, B. Beneš, R. Měch, D. G. Aliaga, and P. Křištof. Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum*, 29(2):665–674, 2010. 3