ShaderTransformer : Predicting Shader Quality via One-shot Embedding for Fast Simplification

Yuchi Huo*
huo.yuchi.sc@gmail.com
State Key Lab of CAD&CG, Zhejiang University
Zhejiang Lab
China

Shi Li*
lishi0927@zju.edu.cn
State Key Lab of CAD&CG, Zhejiang University
China

Xu Chen
x.chen@zju.edu.cn
State Key Lab of CAD&CG, Zhejiang University
China

Yazhen Yuan
835652868@qq.com
Game Engine Center, CROS, Tencent
China

Wenting Zheng
wtzheng@cad.zju.edu.cn
State Key Lab of CAD&CG, Zhejiang University
China

Hai Lin
lin@cad.zju.edu.cn
State Key Lab of CAD&CG, Zhejiang University
China

Rui Wang
ruiwang@zju.edu.cn
State Key Lab of CAD&CG, Zhejiang University
China

Hujun Bao†
bao@cad.zju.edu.cn
State Key Lab of CAD&CG, Zhejiang University
China

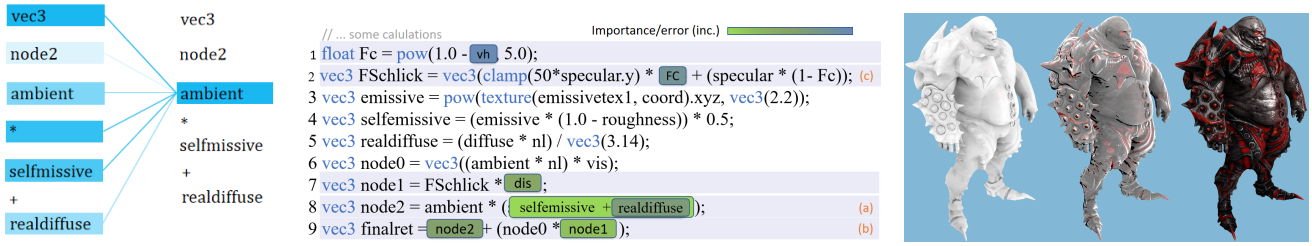

Figure 1 : Based on the NLP technique, our proposed ShaderTransformer can learn complex attention between shader codes (left) to reveal their contributions to shading quality. Once trained with different scene configurations, the network can predict the influence of shader expressions on the final rendering result under different scene configurations. We highlight the top 7 important expressions predicted by our method (center) and show some of the rendering results after simplifying these expressions(right).

RÉSUMÉ

Given specific scene configurations and target functions, automatic shader simplification searches for the best simplified shader variant from an optimization space with many candidates. Although various speedup methods have been proposed, there is still a costly render-and-evaluate process to obtain variant's performance and quality, especially when the scene changes.

In this paper, we present a deep learning-based framework for predicting a shader's simplification space, where the shader's variants can be embedded into a metric space all at once for efficient quality evaluation. The novel framework allows the one-shot embedding of a space rather than a single instance. In addition, the simplification errors can be interpreted by mutual attention between shader fragments, presenting an informative focus-aware simplification framework that can assist experts in optimizing the codes. The results show that the new framework achieves significant speedup compared with existing search approaches. The focus-aware simplification framework reveals a new possibility of interpreting shaders for various applications.

*. Denotes equal contribution.
†. Denotes corresponding author.

Authors' addresses: Yuchi Huo*, huo.yuchi.sc@gmail.com, State Key Lab of CAD&CG, Zhejiang University and Zhejiang Lab, China; Shi Li*, lishi0927@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Xu Chen, x.chen@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Yazhen Yuan, 835652868@qq.com, Game Engine Center, CROS, Tencent, China; Wenting Zheng, wtzheng@cad.zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Hai Lin, lin@cad.zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Rui Wang, ruiwang@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Hujun Bao†, bao@cad.zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China.

2022. 0730-0301/2022/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS CONCEPTS

• **Computing methodologies → Rendering**.

KEYWORDS

Transformer, Shader Simplification, Real-time Rendering

ACM Reference Format:

Yuchi Huo*, Shi Li*, Xu Chen, Yazhen Yuan, Wenting Zheng, Hai Lin, Rui Wang, and Hujun Bao†. 2022. ShaderTransformer : Predicting Shader Quality via One-shot Embedding for Fast Simplification. *ACM Trans. Graph.* 1, 1 (May 2022), 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The increasing flexibility of modern graphics hardware allows the rendering of attractive visual effects via complex shaders. Automated shader simplification is a promising solution for meeting real-time requirements in various scene complexities. Several works [He et al. 2015; Olano et al. 2003; Pellacini 2005; Sitthiamorn et al. 2011; Wang et al. 2014; Yuan et al. 2018] have proposed to automatically simplify shaders. These works proposed simplification rules including removing operations [Olano et al. 2003], moving fragment shader codes to the vertex/tessellation shader stage [Wang et al. 2014], and moving fragment shader codes to the parameter stage [He et al. 2015]. Applying these rules to the original shader produces a large set of simplified shader variants. Then, searching schemes, such as genetic programming (GP), were introduced to efficiently find the optimal shader variants on the Pareto frontier defined by metrics such as rendering performance and visual quality. However, the simplification framework remains a time-consuming process, mostly because of the cost of the process of rendering and evaluating many shader variants. Classic offline methods [He et al. 2015; Pellacini 2005; Sitthiamorn et al. 2011; Wang et al. 2014] seek the average best by rendering a shader variant in multiple scene configurations and averaging the result, which makes existing solutions miss optimal solutions, take a considerable amount of time, and consume a large space to cache the results. Subsequent works have proposed replacing the evaluation by predicting the rendering performance and visual quality through rendering spare variant samples [He et al. 2015; Yuan et al. 2018]. However, they still need to render example scene configurations multiple times to guide the search. The shader evaluation is often conducted one by one and takes up a large portion of the search time.

The challenge is that although rendering performance can be effectively predicted with a small cost, the visual quality directly depends on the multimodal inputs consisting of shade codes and scene configurations. For example, a simplified shader that removes AO terms may yield good quality in a flat scene but may fail on bump surfaces.

To address such a challenge, in this paper, we consider two modifications to provide a better evaluation of shader candidates. First, we decouple the scene configurations with the shader codes. Therefore, we do not need to conduct the render-and-evaluate process while the scene changes. Second, we want to evaluate the qualities of simplified variants all at once and thus avoid costly one-by-one evaluation.

However, analyzing shaders is a complicated multimodal challenge that involves programming languages, mathematical functions, and 3D inputs. Recently, the natural language processing (NLP) community has been exhilarated by the significant success of the Transformer [Vaswani et al. 2017b] structure. The new structure can efficiently capture contextual information from the entire sequence rather than one locality. We find that this property is suitable for finding the distant relationship between variables in the shader code. Therefore we propose a Transformer that takes shader inputs such as G-buffers and uniform parameters as conditions. Unlike common neural network structures, the transformer's output is not specific values but represents the original shader's simplification space. We can easily embed shader variants as vectors by manipulating the output attention cache. Then the visual errors are naturally represented by the distances between the embeddings of the original shader and its variants.

Specifically, we first train a model from scratch for each shader. At run time, a trained model takes arbitrary shader inputs to predict a simplification space, where we can analytically query shader variants' visual quality. Using this process to replace the existing render-and-evaluate process significantly improves the search speed and accuracy. In contrast, GP cannot generalize to new configurations and needs to re-evaluate each object's variants to avoid over-averaging. Thus, it takes a lot of time for applications like games with thousands of different objects.

Our contributions are summarized as follows :

- A multimodal conditional Transformer that takes shader inputs as conditions to embed the shader variants into a simplification space according to visual quality.
- A fast automatic shader simplification framework that replaces the render-and-evaluate process with a deep learning-based embedding mechanism.
- A deferred query mechanism that enables the one-shot embedding of the entire space with one neural network inference. It reduces the embedding overhead by several orders.

2 RELATED WORK

Shader Simplification : Shader simplification can be regarded as the approximation of signals on the surface. Technically, the computations of surface signals can be approximated and simplified by removing operations or reducing sampling rates. Texture removal [Olano et al. 2003] was introduced by Olano et al. [Olano et al. 2003] for lossy simplification and Pellacini extended simplification rules with the removal of binary operations or loop expressions [Pellacini 2005],

The core idea of reducing sampling rates is to move per-fragment shader terms to other stages and then use these terms in the fragment shader. Olano et al. [Olano et al. 2003] and Pellacini [Pellacini 2005] proposed to prebake some per-fragment shader terms as textures or constants. Wang et al. [Wang et al. 2014] moved these terms to vertex or tessellation shaders. He et al. [He et al. 2015] moved per-fragment or per-vertex operations into the parameter shader stage.

Besides, searching is another step of shader simplification. Pitchaya Sitthi-amorn et al. [Sitthiamorn et al. 2011] proposed a genetic programming-based shader simplification scheme to select

optimal shaders that balance performance and image quality. He et al. [He et al. 2016] presented a new shading language and compiler framework named Spire that facilitates rapid exploration of shader optimization choices. Yuan et al. [Yuan et al. 2018] proposed a parallel runtime algorithm to efficiently search in the simplification dependency graph for optimal simplified shaders.

Beyond that, some extended applications have been explored. Song et al. [Song et al. 2011] simplified both geometry and shaders. They adopted a progressive mesh to obtain an appropriate geometric representation and employed error control to choose the levels of the shader and geometry. Lewis and Michael [Crawford and O'Boyle 2018] focused on the impact of compiler optimization. They explored common features of graphics shaders and examined the impact and applicability of common optimizations.

Code Processing : Considerable neural network models have been proposed for automatic code summarization, documentation, retrieval, and generation. Several representations have been introduced to represent programs, for example, a sequence of tokens [Bhoopchand et al. 2016; Hindle et al. 2012] and the syntax tree structure of codes [Raychev et al. 2016; **?**]. Recurrent neural network(RNN) architectures, including bidirectional recurrent neural networks (BRNNs) [Cho et al. 2014], LSTMs [Alon et al. 2018], and tree-structured LSTMs [Tai et al. 2015], have been widely adopted to encode a sequence of tokens or the syntax tree structure of codes. In general, the syntax tree structure of codes can also be regarded as a graph, and therefore graph neural networks (GNNs) [Allamanis et al. 2017] have been applied to represent codes in the network. Cummins et al. [Cummins et al. 2017] developed a network that learns heuristics over raw code for simultaneous code representation and optimization. Chen et al. [Chen et al. 2018] introduced a learning-based framework that learns cost models to guide the search of tensor operator implementations over program variants for deep learning workload optimization. In the computer graphics community, a novel framework has been proposed to automatically schedule Halide programs for high-performance image processing and deep learning [Adams et al. 2019]. It outperforms human experts and produces schedules that are almost twice as fast as the existing Halide autoscheduler.

Transformer : The Transformer architecture and self-attention models have been widely used in natural language processing (NLP) and have revolutionized various NLP applications [Dai et al. 2019; Kenton and Toutanova 2019; Vaswani et al. 2017a; Wu et al. 2018]. These works also inspire the evolution of self-attention-based image processing neural networks [Dosovitskiy et al. 2020; Hu et al. 2019; Parmar et al. 2018; Ramachandran et al. 2019; Zhao et al. 2020]. The key to success is the self-attention model that enables the network to capture contextual information from the entire sequence rather than locality. Technically, each element actively queries the correlation with every other element regardless of distance, forming dynamic connections depending on the inputs. While classic shader code process techniques have trouble building connections between distant variable pairs, the Transformer's capability of learning contextual information from the long sequence is valuable for analyzing shader codes.

Metric Learning : Metric learning analyzes the similarity between data to embed data into a latent manifold space. After the embedding, high-dimensional data can be handily applied in various applications, such as image recognition [Hadsell et al. 2006] and image retrieval [Wang et al. 2017]. For computer graphics, a recent work adapts contrastive learning to explore the coherent information within the high-dimensional path space [Cho et al. 2021]. A recent survey covers more details [Kaya and Bilge 2019]. This paper focuses on embedding shader variants into a manifold with distances following simplification errors. Unlike common embedding techniques that process single instances one by one, we propose one-shot embedding to efficiently embed the entire simplification space, i.e., the set of many shader variants, with only one neural network inference. The one-shot embedding technique reduces the embedding cost by several orders of magnitude.

3 PROBLEM STATEMENT

At a high level, the goal of shader simplification is to find a sequence of shader variants that take less time to render while preserving more visual quality than other variants. The classic shader simplification approach relies on an offline process to generate a sequence of simplified shader variants from an original shader s_o. Figure 2 illustrates the flowchart of such a framework. The original shader code is usually converted to an abstract syntax tree (AST). Different shader simplification rules may be applied to generate a set of simplified shader variants. The generated variants can be further simplified, thus generating more variants in the next iteration. Exhaustive exploration of all possible variants is costly, and previous methods conduct the render-and-evaluate process to obtain variants' rendering quality and performance, then choose a smaller number of representative variants for further simplification. For example, GP[Sitthiamorn et al. 2011; Wang et al. 2014] or greedy search [He et al. 2015] selects variants at the Pareto frontier, and the online method [Yuan et al. 2018] selects variants at the cluster center in preprocessing and prioritizes them using runtime search.

To evaluate a simplified shader variant s_v, and provide a specific scene with multiple shader inputs, e.g., geometry, uniforms, and textures, the engine renders a scene, then acquires a rendered image, and the computation cost $t(s_v)$. The rendering quality can be measured by comparing the rendered image with the ground truth (GT) image rendered by the original shader s_o :

$$\varepsilon(s_v, u) = \|I(s_v, u) - I(s_o, u)\|_2 , \tag{1}$$

where u represents generalized shader inputs, such as uniform parameters and geometric attributes of the given scene configuration, and I denotes the rendered image. Then, shader simplification is formulated as a multi-objective problem in the two-dimensional cost space (ε, t). Finally, Pareto-optimal solutions are used to seek better shader variants.

In general, the rendering quality is a function of the shader variant s_v and shader inputs u. To obtain better generalization to scene changes in runtime, offline methods sample different scene configurations to acquire the average rendering quality ε and computational cost t. However, each sampling means rendering the scene one time, thereby evaluating multiple shader variants with multiple scene configurations making the entire simplification process time-consuming.

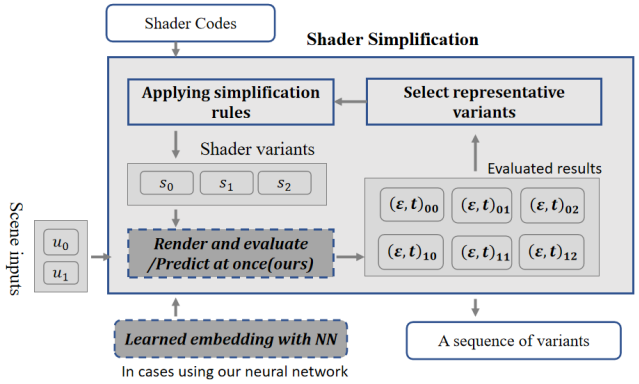

Figure 2 : Shader simplification framework, we use our learned embedding to accurately predict multiple shader variants' qualities all at once.

In our paper, we use deep learning to efficiently evaluate the rendering quality of the shader variants by training a neural network to learn ε. We introduce a conditional transformer to predict a novel simplification space of the original shader. In the simplification space, shader variants are embedded as vectors, and their quality can be handily measured with distances all at once. We consider that this architecture is parallel to a specific simplification framework and can be greatly beneficial to both online and offline methods.

4 NETWORK

The design of the network faces several challenges. First, the evaluation is a multimodal function that takes both shader inputs and shader codes as inputs. Inspired by NLP, we propose a conditional Transformer structure to process shader codes and communicate information between multimodal inputs. The structure first encodes shader inputs via a PointNet-like branch and then inserts the transformer's encoding. Second, there are countless shader variants of the original shader, and it is inefficient to infer the shader variants one by one. We prefer evaluating a space rather than a single shader variant at one network inference. Therefore, we propose a simplification space that can embed the shader variants with rendering quality. Instead of directly inferring specific values, the network infers an attention cache, which enables the one-shot embedding of the entire space rather than a single instance.

The overall pipeline is shown in Figure 3. Given original shader s_o and scene configuration u, the target is to predict the rendering quality of various shader variants s_v (Eq. 1). The backbone of the network structure is the shader transformer, which processes shader codes like NLP. Because scene configurations can determine the rendered content of a shader, it is important to train the network to generate scene-dependent results. Here, we use the input encoder to encode the scene as a condition for the context encoder. The node encoder, node decoder, key, query, and value blocks are used to convert multimodal data, such as shader codes, implicit vectors, and simplification space representations.

In general, we parse the original shader's codes into nodes. Then, the MLP-based shader encoder converts the codes to latent vectors. Next, the context encoder learns the relationship between shader nodes to predict a global context vector (CV). Additionally, the shader inputs of the scene configuration are encoded as a condition of the context encoder. Finally, the node decoder combines the global CV and node latent vectors into a node vector. Three small blocks, key, query, and value convert the node vectors into an attention cache representing the shader variant's embedding in the simplification space. The rendering quality can be directly extracted from the simplification space by measuring the distance between the embedding of the shader variant and the original shader. In the following, we discuss each specific design. The detailed network structure is in the supplementary document.

4.1 Node Encoder

We apply the general NLP method to encode our shader codes. First, we still use an abstract syntax tree to represent the codes, and each symbol in the codes is regarded as an intermediate node. Then, we traverse the tree nodes in a depth-first order to encode the shader codes as sequences of words. Subsequently, we embed the words into 128-dimensional vectors. As in the common Transformer neural network [Vaswani et al. 2017b], each embedded vector and its position encoding are fed into the MLP-based node encoder to generate a shader node m_x.

4.2 Input Encoder

Typical shader inputs consist of vertex attributes (position, normal, uv coordinates, etc.), shader uniforms (view or light position, etc.) and textures. They have various dimensions and are not suitable for networks to process. Our network focuses on the screen space error, so following a common technique like G-buffer [Deering et al. 1988], we generate a buffer consisting of the attributes from rasterization and shader uniforms, such as "worldpos" or "lightdir" buffers in the common shader.

Given a scene configuration, the input encoder processes the shader inputs to a vector as the condition of the shader transfer. The structure of the input encoder is based on PointNet [Qi et al. 2017]. Because the fragment shader is a function of the shader inputs on a single pixel, we use PointNet rather than CNN to prevent introducing inaccurate correlation between pixels. This technical choice significantly improves the generalization capability of the input encoder. The uniform parameters are shared by all pixels. We simply stack them on the end of the G-buffer to form a high-dimensional point as network inputs.

4.3 Context Encoder

The context encoder is the backbone of the network with two functions. First, it learns the relationship between shader nodes. The Transformer structure is a state-of-the-art NLP model that has recently attracted significant attention [Vaswani et al. 2017b]. It can efficiently explore the long-distance relationship between nodes and preserve the information of the orders simultaneously. This feature benefits the exploration of the contextual connections of variables in the shader codes. Second, it learns the correlation between multimodal information. The shader inputs are image-space features, but the shade nodes are language information. Considering the application scenario, we treat the encoded shader inputs as

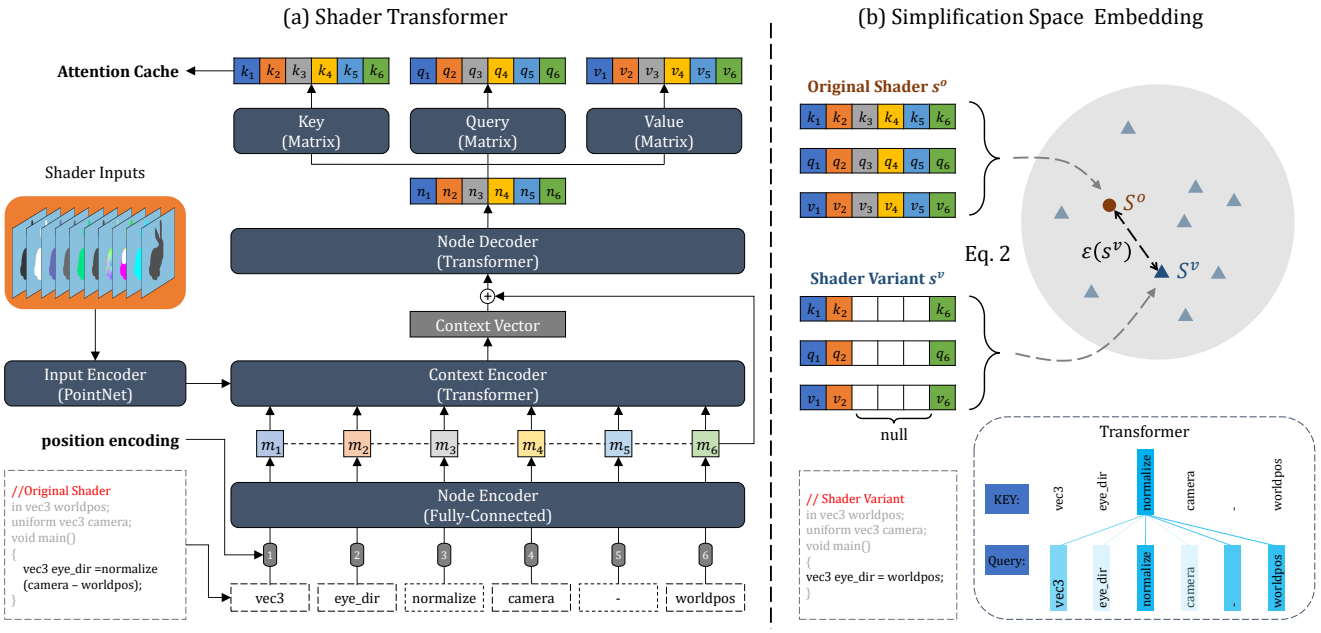

Figure 3 : Overview of the Shader Transformer and the simplification space.

a condition of the context encoder. Therefore, once the network is trained for the original shader, we can feed dynamic shader inputs into the network for adaptive simplification. The final output of the context encoder is a context vector (CV) that contains the global information of the shader.

4.4 Node Decoder

The decoding part of the network includes the Transformer-based node decoder and three lightweight blocks. The CV is in contact with the shader nodes as the inputs of the node decoder. Intuitively, the network combines global information (CV) with local information (shader node) to predict node embedding n_i. The trainable key, query, and value blocks, respectively convert n_i into corresponding attention elements $k_i \equiv Key(n_i)$, $q_i \equiv Query(n_i)$, and $v_i \equiv Value(n_i)$. Here k_i and q_i are scalars, and v_i is a vector. These elements jointly represent the simplification space as discussed next.

4.5 Deferred Query

Inspired by the attention mechanism, we design a deferred query mechanism to represent the simplification space all at once. As shown in Figure 3, we cache the network outputs and perform the query of each shader variant's embedding analytically. Technically, the final embedding S_v of the shader variant is computed as :

$$
\begin{aligned}
S_v &= softmax(\frac{Q \times K^T}{\sqrt{d_k}}) \cdot V, \\
Q &= [Query(n_1), Query(n_2), ..., Query(n_v)], \\
V &= [Value(n_1), Value(n_2), ..., Value(n_v)], \\
K &= [Key(n_1), Key(n_2), ..., Key(n_v)],
\end{aligned} \tag{2}
$$

where $Key(\cdot)$, $Query(\cdot)$, and $Value(\cdot)$ represent the three corresponding network blocks, v denotes the number of nodes of the original shader, and $d_k = 128$ is a normalization term [Kenton and Toutanova 2019]. If a node n_i has been deleted from a simplified shader variant, we use a dummy vector *null* to replace n_i in the equation to embed the variant. As shown in Figure 3 (b), we use the full attention cache to analytically embed original shader s_o. However, for simplified shader s_v, the cache elements of the corresponding nodes are replaced with a dummy constant vector *null*.

4.6 Loss Function

Inspired by deep metric learning, the embedding is trained by a loss function based on the shading error of the shader variant. Specifically, the loss function is

$$
Loss(s_v, u) = \| \frac{\hat{\varepsilon}(s_v, u) - \varepsilon(s_v, u)}{\varepsilon(s_v, u) + \delta} \|_2 + \lambda \|\omega\|_2. \tag{3}
$$

We use relative L2 loss in our loss function because the application in shader simplification will be more interested in shader variants with minor difference values. We add hyperparameter $\delta = 1e^{-}4$ to avoid zero in the division. Note that $\hat{\varepsilon}(s_v, u)$ is the predicted difference value of our neural network, and $\varepsilon(s_v, u)$ is the GT difference value. We also apply L2 regularization to our training parameters in our training. The regularization weight is $\lambda = 0.1$.

5 NETWORK TRAINING

5.1 Shader Simplification Rules

A simplification algorithm would choose suitable simplification rules, as described in Sec.2, to simplify the original shader. Currently, we focus on rules that do not change shader inputs. Therefore,

other rules, such as code lift [He et al. 2015; Wang et al. 2014], are considered as future extensions. For now, our system supports two rules :

(a) Expression reduction : Expression reduction can efficiently eliminate algebraic operations and has been used widely in several shader simplification systems. We adopt two expression reduction rules proposed by Pellacini, but discard the rules requiring runtime data and information. More precisely, let us consider the binary operator $a \otimes b$; after a reduction, we have two simplified expressions, $a \otimes b \rightarrow a$ and $a \otimes b \rightarrow b$. We also add if-removing optimization, and we randomly choose one of the branches in the if-else statement.

(b) Expression replacement : Expression replacement is a new effective rule that replaces statements with other expressions or values that have been previously computed. For example, the variable "NdotH" in shader codes can be replaced with "NdotL", thus saving extra dot instructions.

5.2 Training Data Generation

Our approach takes four steps to generate data. With the original shaders, we first generate simplified shader variants according to the simplification rules in Section 5.1. Second, we randomly choose valid scene configurations and render images of the original shader and simplified variants. Buffers and shading results are recorded. Then, we parse each shader and extract intermediate nodes. Finally, we pack buffers, the mean-square error of shading results, and intermediate nodes of both the original and simplified shaders as one training sample.

Example Scene : Models with different scales of shader inputs are included in the sample scenes. Ten models, such as a dragon and a bunny, are used in our early experiments. Then, we collect 25 extra additional CAD models from ModelNet10 [Wu et al. 2015]. After showing promising results in a single model, we further extend our database to include five large indoor scenes, such as Sponza and breakfast room. The complex materials and textures that come with these scenes further validate our approach with complex shaders. In summary, we collect about 40 different scenes/models, 32 of them are used for training and 8 for testing.

Data Generation : Camera position and light condition are sampled around the object at different distances to generate sufficient random data while maintaining meaningful rendering results. For uniform parameters, we also randomly sample the values in a physically valid range. For example, roughness is sampled within [0,1]. Different meshes and associated textures are provided to introduce diverse shader input data.

Because variants can share the shader input buffer during training, for a particular scene/uniform sample, we only need to generate a shader input buffer once. We use the abstract syntax tree (AST) to automatically transform the code to support deferred rendering. In this way, all shader variants can share the same shader input buffer so that a great amount of rasterization costs are saved.

5.3 Training Process

We trained our network on one machine with an NVIDIA 2080Ti GPU (11 GB video memory). The size of the training data highly depends on shader codes. Table.1 shows some statistics of our experiments. For a shader with 900 variants and 1,000 shader inputs, it takes approximately 24 hours(10,000 epochs) to converge.

Optimizer and Regularization : We use the Adam optimizer [Kingma and Ba 2014] with a learning rate of $1 * 10^{-5}$ and employ two types of regularization during training : dropout and layer normalization. Similar to the Transformer [Vaswani et al. 2017b], we apply dropout and layer normalization in each sublayer. In addition, we use dropout in our multiple perception network to encode context.

Importance Sampling : We first sample thousands of scene configurations, then apply importance sampling to reduce the number of scene configurations to a trainable size. For each scene configuration, we compute the variance of all variants' rendering errors. Then we use variance as a probability density function to sample 400 scene configurations. In addition, we uniformly sample 600 scene configurations, producing a training set with $1,000$ scene configurations.

6 EVALUATION

We use glslang to parse the shader code into AST, on which simplification rules can be applied and intermediate nodes can be easily extracted. The network is trained on TensorFlow with a 256×256 shader input buffer. All results in this section are measured using our test set that consists of scenes outside the training set. The experiments are conducted using a PC with an Intel Xeon Silver 4110 CPU and an NVIDIA GeForce GTX 2080Ti GPU.

As shown in Figure 2, we implement the classic GP-based offline shader simplification framework. The time-consuming render-and-evaluate stage is replaced by the deferred query of the attention cache. Due to the parallel nature of neural networks, a large number of shader variants can be evaluated simultaneously. We modify the heuristic performance model presented in [He et al. 2015] by deleting the vertex stage cost using a simple performance model that counts float operations in the fragment shader for each variant.

We conduct several experiments to prove that our one-shot embedding can boost the optimization process by replacing the costly render-and-evaluate process. We believe our embedding can be beneficial to other heuristic-based search methods.

6.1 Embedding Accuracy

We use a prediction error (E) to illustrate the average distance between our predicted error and the measured error. As shown in Table 1, most of the training errors are less than 10^{-4}, but the test error of complex shaders with layers of textures are significantly larger than basic BRDF shaders due to scene complexity. In Figure 5, we show the results of each variants' predicted and evaluated error. These results validate our neural network which has perfect overall effects in estimating visual errors and is accurate enough to provide a useful guide for shader simplification.

Moreover, we show an experiment in Figure 4 to verify that the network can generalize to different scene configurations : left is a glossy dragon and right is a diffuse box. Because our embedding is conditional to the shader inputs, the evaluated and predicted errors are closely matched in both scenes. Closely examining the shader codes, we see :

origin : float diff = 0.5 * NdotL ; gl_FragColor = diff + spec ;

Table 1 : Results Analysis. Various shaders with different inputs are tested in our experiments. The results show that our method can give accurate predictions in new scenes (row ϵ_{test}). After replacing the render-and-evaluate process with our network prediction, the shader simplification consumes considerably less time than traditional methods (row T_{our}, T_{infer} and T_{render} are measured in minutes). T_{infer} reports network usage time to directly to infer variant quality without our deferred query scheme.

	Basic BRDF				Layers of textures		
Shader	*Blinn-Phong*	*GGX*	*Cook-Torrance*	*SSS*	*Indoor*	*Imrod*	*Couch*
Variants#	97	891	536	1383	173	1217	1160
$\epsilon_{train} * 10^{-5}$	0.5	5.4	17.6	6.9	6.0	3.0	3.0
$\epsilon_{test} * 10^{-5}$	1.8	5.9	10.9	1.7	195.7	1507	34.1
T_{our}	0.48	0.82	0.68	1.25	0.52	1.13	1.05
T_{infer}	1.55	2.12	1.85	3.96	1.58	3.85	3.52
T_{render}	8.2	28.8	36.8	10.7	23.4	205.2	109.9

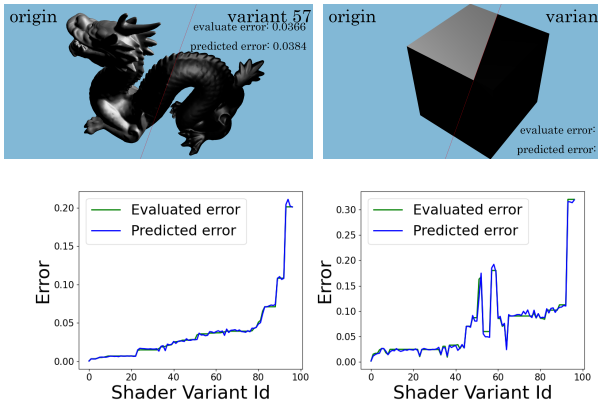

Figure 4 : Bottom : The predicted and evaluated error of a Blinn-Phong shader's variants in ascending order; under a new scene configuration, the same shader variant has different rendering quality, which is also well predicted by our method. Top : The visual comparison between the original and simplified shader in two sample scenes.

Vid 57 : float diff = 0.5 * NdotL; gl_FragColor = spec;
Vid 58 : float diff = 0.5; gl_FragColor = spec.

Since the $diffuse$ term is removed in variant 57, the evaluated error in the diffuse box scene is disproportionately larger than in the glossy dragon scene. In variant 58, the variable $NdotL$ is also removed, but the rendered results remain the same as in variant 57. Our network captures such relations and gives a close prediction.

6.2 GP-based Shader Simplification Comparison

We modify the classic genetic programming (GP) method by replacing the evaluation stage. Note that the original paper [Sitthi-amorn et al. 2011] introduces complex rendering methods to speed up the evaluation. However, in our implementation, the rendering quality of multiple variants can be queried all at once, resulting in an average query time of 0.7 ms for one variant. In contrast, the actual rendering of a small patch to evaluate the quality costs 80-120 ms. As summarized in Table 1, GP requires up to hours in some complex scenes to render multiple times for each simplified shader variant. Our method requires approximately only one minute.

The Pareto frontier is significant for shader simplification algorithms. In Figure 6, we show that the Pareto frontier found by our modified simplification algorithm closely matches the classic method using the evaluated error in different scene settings except on one point in the *Cook-Torrance* shader. More results can be found in the supplementary document.

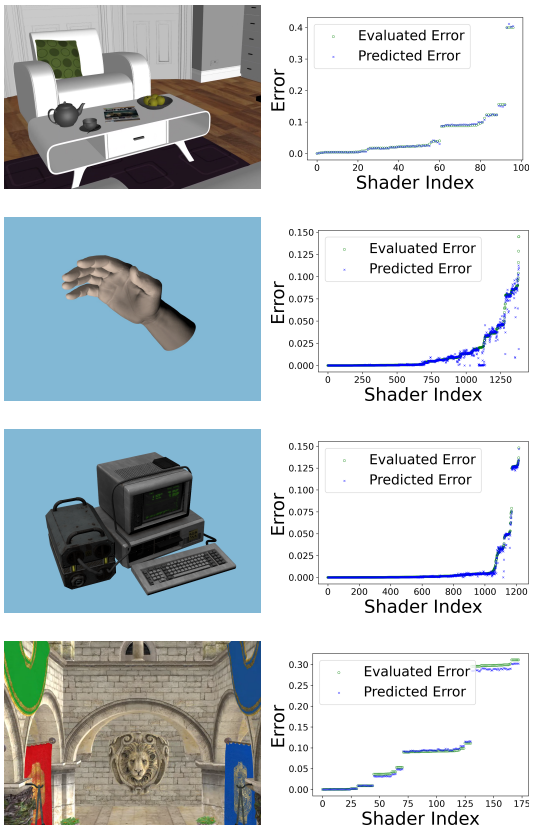

Figure 5 : Comparing our predicted error with the measured error in different shaders and scenes. The shaders are *Blinn-Phong*, *Subsurface Scattering*, *Imrod*, and *Indoor*.

6.3 Ablation Study on One-shot Embedding

The core idea of one-shot embedding is the use of one inference to predict multiple variants' qualities via the deferred query scheme. In Table 1, we present the results of an ablation study that disables the deferred query scheme and reports the prediction time in T_{infer}. Note that in our current implementation, the deferred query process is not fully optimized by parallel computing. Therefore the measured speedup is smaller than the theoretical speedup.

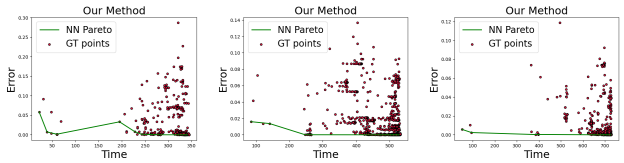

Figure 6 : Pareto frontier of *Cook-Torrance* (left), *Subsurface Scattering (center), and Couch (right) shaders based on the error predicted by the network.*

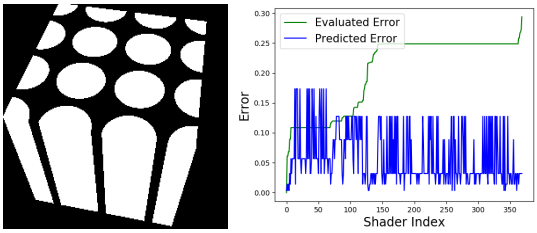

Figure 7 : Our network gives poor prediction on shaders that produce discontinuous shading results, such as the procedural *Circle* shader shown on the left.

6.4 Shader Expression Importance

In practice, a trial and error simplification process guided by an expert is commonly used. However, it is usually difficult for a graphics expert to tell which statement/expression has contributed most to the overall rendering quality. Fortunately, our embedding approach can give a direct mapping between the shader code and the rendering quality condition to the shader inputs. We can now even back-trace a rendering error to the expression and variables, thus gaining a powerful insight into the quality importance of the expression shown in Figure 1. Specifically, to obtain the rendering error of eliminating an expression or statement, we can skip the tedious shader compiler and rendering by simply setting corresponding nodes with dummy constant vector *null*, and the query can be performed in batch. After all the expressions have been queried, we divide the queried error by $\epsilon = ||0 - I(S_0, u)||_2$, namely, the error of a black image, to obtain normalized importance.

7 DISCUSSION AND LIMITATIONS

Our neural network performs poorly with shaders that generate high-frequency and discontinuous shading signals. Figure 7 shows a failure case using the procedure shader *Circle* [Li et al. 2020; Yang and Barnes 2018] that conducts $fract$ and *sign* functions on a surface position, which produces white circles in black cubes with hard boundaries.

Our neural network has two main limitations that can be studied in the future. First, it can deal with limited shader inputs and now only supports less than 32 float parameters, including textures and uniform parameters. There is a need to expand the size of input encoder component. Second, our neural network still only focuses on fragment shaders, and some useful simplification rules such as move to vertex stage rules [Wang et al. 2014] or move to parameter stage rules [He et al. 2015] require representing the graphics pipeline using a neural network. We think this will be an interesting topic for graphics studies.

8 CONCLUSION

To speed up the time-consuming render-and-evaluate process, we present a deep learning-based framework based on a conditional Transformer to predict shader variants' qualities. First, our neural network encodes shader inputs via a PointNet-like branch to communicate information between multimodal inputs. We also propose a simplification space that can embed the shader variants with rendering quality. Instead of directly inferring specific values, the network infers an attention cache, which enables one-shot embedding of the entire space rather than a single instance. We conduct several experiments to prove that our one-shot embedding can boost the optimization by replacing the costly render-and-evaluate process.

Experiments show that our method produces accurate embedding and Pareto frontier results. Most remarkably, the search time improves by approximately 350 times. Finally, to the best of our knowledge, this is the first time a transformer has been introduced to analyze shader codes.

One significance of our paper is its attempt to explore correlations within rendering processes using intelligent techniques. An advanced neural network can help our rendering system to adaptively choose a suitable level of detail and balance rendering quality and performance. Using a neural network to accelerate the rendering pipeline will be a worthwhile topic of study.Secondly, recently Program Traces [Yang et al. 2021] have provided a novel direction for learning for shaders. We believe that including some intermediate program traces as inputs will improve our precision and enlarge our application range.Moreover, combining natural language processing with compute graphics already has many applications, such as image captions, but these applications are still simple, and we can explore more sophisticated ones. Finally, a neural network can make the code easy to understand and help turn our idea into reality. Therefore, we also expect more works to be done on this topic.

ACKNOWLEDGMENTS

This work was supported in part by Key R&D Program of Zhejiang Province (No. 2022C01025), NSFC (No. 61872319), the Fundamental Research Funds for the Central Universities, Zhejiang Lab (121005-PI2101), and Information Technology Center and State Key Lab of CAD&CG, Zhejiang University. This work has been integrated in the RaysEngine project.

RÉFÉRENCES

Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, et al. 2019. Learning to optimize halide with tree search and random programs. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–12.

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2017. Learning to Represent Programs with Graphs. arXiv:1711.00740 [cs.LG]

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2018. code2seq : Generating Sequences from Structured Representations of Code. *arXiv : Learning* (2018).

Avishkar Bhoopchand, Tim Rocktäschel, Earl Barr, and Sebastian Riedel. 2016. Learning Python Code Suggestion with a Sparse Pointer Network. (2016). http://arxiv.org/abs/1611.08307

Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. Learning to optimize tensor programs. *Advances in Neural Information Processing Systems* 31 (2018).

In-Young Cho, Yuchi Huo, and Sung-Eui Yoon. 2021. Weakly-Supervised Contrastive Learning in Path Manifold for Monte Carlo Image Reconstruction. *ACM Trans. Graph.* 40, 4, Article 38 (jul 2021), 14 pages. https://doi.org/10.1145/3450626.3459876

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation : Encoder-Decoder Approaches. arXiv:1409.1259 [cs.CL]

L. Crawford and M. O'Boyle. 2018. A Cross-platform Evaluation of Graphics Shader Compiler Optimization. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. 2017. End-to-end deep learning of optimization heuristics. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 219–232.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. 2019. Transformer-XL : Attentive Language Models beyond a Fixed-Length Context. In *ACL (1)*.

Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. 1988. The triangle processor and normal vector shader : a VLSI system for high performance graphics. *Acm siggraph computer graphics* 22, 4 (1988), 21–30.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality Reduction by Learning an Invariant Mapping. In *CVPR*.

Yong He, Tim Foley, and Kayvon Fatahalian. 2016. A System for Rapid Exploration of Shader Optimization Choices. (2016).

Yong He, Tim Foley, Natalya Tatarchuk, and Kayvon Fatahalian. 2015. A system for rapid, automatic shader level-of-detail. *Acm Transactions on Graphics* 34, 6 (2015), 1–12.

Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the Naturalness of Software. In *Proceedings of the 34th International Conference on Software Engineering*.

Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. 2019. Local relation networks for image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3464–3473.

Mahmut Kaya and Hasan Şakir Bilge. 2019. Deep metric learning : A survey. *Symmetry* 11, 9 (2019), 1066.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

Diederik P Kingma and Jimmy Ba. 2014. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980* (2014).

Shi Li, Rui Wang, Yuchi Huo, Wenting Zheng, Wei Hua, and Hujun Bao. 2020. Automatic Band-Limited Approximation of Shaders Using Mean-Variance Statistics in Clamped Domain. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 181–192.

]maddison14 Chris Maddison and Daniel Tarlow. [n. d.]. Structured Generative Models of Natural Source Code. In *Proceedings of the 31st International Conference on Machine Learning*. PMLR. http://proceedings.mlr.press/v32/maddison14.html

Marc Olano, Bob Kuehne, and Maryann Simmons. 2003. Automatic shader level of detail. In *Acm Siggraph/eurographics Conference on Graphics Hardware*.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *International Conference on Machine Learning*. PMLR, 4055–4064.

Fabio Pellacini. 2005. User-Configurable Automatic Shader Simplification. *ACM Trans. Graph.* (2005).

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet : Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 652–660.

Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. 2019. Stand-alone self-attention in vision models. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 68–80.

Veselin Raychev, Pavol Bielik, and Martin Vechev. 2016. Probabilistic Model for Code with Decision Trees. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Amsterdam, Netherlands) *(OOPSLA 2016)*.

Pitchaya Sitthiamorn, Nicholas Modly, Westley Weimer, and Jason Lawrence. 2011. Genetic programming for shader simplification. (2011).

X. Song, C. Tu, and Y. Xu. 2011. An Adaptive Method for Shader Simplification. In *2013 International Conference on Virtual Reality and Visualization*. IEEE Computer Society, Los Alamitos, CA, USA, 46–51.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. arXiv:1503.00075 [cs.CL]

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017a. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017b. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Jian Wang, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin. 2017. Deep Metric Learning with Angular Loss. In *IEEE International Conference on Computer Vision, ICCV 2017*. IEEE Computer Society, Venice, Italy, 2612–2620.

Rui Wang, Xianjin Yang, Yazhen Yuan, Wei Chen, Kavita Bala, and Hujun Bao. 2014. Automatic Shader Simplification Using Surface Signal Approximation. (2014).

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2018. Pay Less Attention with Lightweight and Dynamic Convolutions. In *International Conference on Learning Representations*.

Null Zhirong Wu, Shuran Song, Aditya Khosla, Null Fisher Yu, Null Linguang Zhang, Null Xiaoou Tang, and Jianxiong Xiao. 2015. 3D ShapeNets : A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yuting Yang and Connelly Barnes. 2018. Approximate program smoothing using mean-variance statistics, with application to procedural shader bandlimiting. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 443–454.

Yuting Yang, Connelly Barnes, and Adam Finkelstein. 2021. Learning from Shader Program Traces. *arXiv preprint arXiv :2102.04533* (2021).

Yazhen Yuan, Rui Wang, Tianlei Hu, and Hujun Bao. 2018. Runtime Shader Simplification via Instant Search in Reduced Optimization Space. *Computer Graphics Forum* (2018).

Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. 2020. Exploring self-attention for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10076–10085.