Automatic Mesh and Shader Level of Detail

Yuzhi Liang, Qi Song, Rui Wang, Yuchi Huo, and Hujun Bao

Abstract—The level of detail (LOD) technique has been widely exploited as a key rendering optimization in many graphics applications. Numerous approaches have been proposed to automatically generate different kinds of LODs, such as geometric LOD or shader LOD. However, none of them have considered simplifying the geometry and shader at the same time. In this paper, we explore the observation that simplifications of geometric and shading details can be combined to provide a greater variety of tradeoffs between performance and quality. We present a new discrete multiresolution representation of objects, which consists of mesh and shader LODs. Each level of the representation could contain both simplified representations of shader and mesh. To create such LODs, we propose two automatic algorithms that pursue the best simplifications of meshes and shaders at adaptively selected distances. The results show that our mesh and shader LOD achieves better performance-quality tradeoffs than prior LOD representations, such as those that only consider simplified meshes or shaders.

Index Terms—Level of detail, geometry simplification, shader optimization, real-time rendering

1 Introduction

The real-time rendering of complex, realistic models is of great interest in a variety of computer graphics applications. Currently, even though graphics hardware has evolved rapidly, the rendering of complex models is still a very challenging task for the latest devices. This tension has inspired a large amount of excellent research attempting to bridge complexity and performance. The level-of-detail (LOD) technique is a widely used method and serves as a key rendering optimization in many applications.

The basic idea of an LOD is to use a less detailed, simplified representation for small, distant, or unimportant portions of models. Different kinds of LODs have been explored to different extents. A geometric LOD with the mesh simplification technique is one of the most well-known representations and has been successfully applied in many applications [1]. In addition, with the development of GPUs, especially with the advancement of shader programming to enhance model appearance fidelity, shader LOD [2], [3] has been proposed to balance performance and rendering quality at distances. Numerous automatic methods have been proposed to create these two kinds of LODs [4] [5] [6] [7] [3] [8] [9]. However, none of these methods have considered the simultaneous simplification of both geometry and shaders.

In this paper, we present a new multiresolution representation of objects, which consists of mesh and shader LODs (Figure 3). Each level of the representation could contain both simplified representations of shaders and meshes. Compared with a traditional geometric LOD [4] [5] [6] [7] or shader LOD [3] [8] [9], our new representation greatly extends the tradeoffs between the performance and the rendering quality; therefore, it achieves better performance at the same quality or better quality at the same performance.

- *Qi Song was with Booming Tech, Hangzhou, China. Other authors were with State Key Lab of CAD&CG, Zhejiang University, China.*
- *Corresponding author: Rui Wang, rwang@cad.zju.edu.cn*

To create such a new LOD representation, we propose two automatic optimization algorithms. The first algorithm simultaneously explores different simplified meshes and simplified shaders to find best-simplified pairs at different distances, i.e., the algorithm simplifies both mesh and shader in an interleaved manner. Specifically, we first simplify the input shader with genetic programming to obtain one generation of shader variants and employ an image-driven mesh simplification method on every shader variant on the Pareto frontier to obtain simplified mesh and shader pairs within an error bound. Then, these shader variants of those pairs with better rendering performance are selected to produce the next generation of shader variants. By iteratively performing the two steps mentioned above, we can obtain a collection of optimized pairs, and the pair with the best performance is finally selected. However, the combination of simplified meshes and shaders at each step greatly enlarges the optimization space, where it becomes too time-consuming to be applied for practical use.

To address this challenge, we then propose an approximate but more efficient algorithm. The second algorithm separates the interleaved optimization of mesh and shader into two steps. In the first step, we perform stand-alone simplifications on the mesh and shader to create two sequences of simplified meshes and shaders. Then, we combine these simplified variants in the second step to adaptively search for the best pairs at different distances. To accelerate the search, we further utilize the monotonicity of quality loss of increasingly simplified meshes and shaders and manage to reduce the search to a 1D search in the 2D space of simplified meshes and shaders. As a result, the second algorithm greatly reduces the computational complexity but achieves similar optimization quality compared with the first optimization algorithm.

Our algorithms are capable of operating on meshes and shaders used in both forward and deferred rendering contexts, and in many cases, the second algorithm produces a good quality LOD in minutes rather than hours. The results show that our mesh and shader LOD

achieves better performance-quality tradeoffs than prior LOD representations, such as those using only simplified meshes [4] [5] [6] [7] or shaders [3] [8] [9].

The main contributions of this paper are as follows:

- a new multiresolution representation of objects, which includes the mesh and shader LODs, with both simplified geometric and shading details;
- an algorithm that interleavingly simplifies both mesh and shader to obtain the optimized pair of simplified meshes and shaders within an error bound at a specific distance;
- an approximate optimization algorithm that combines the simplification of mesh and shader, in particular, utilizes the monotonicity of quality loss of increasingly simplified meshes and shaders, performs 1D search in the 2D space, and generates LODs with desirable performance-quality tradeoffs.

2 Related Work

Our method incorporates ideas from prior mesh or shader simplification work but contributes a new LOD representation that combines a simplified mesh and shader. In the following, we briefly introduce related approaches to mesh or shader simplification and LOD creation.

Mesh Simplification and LOD. The basic principle behind LOD was first introduced by Clark [10]. In these early days, LODs were usually created by hand. With the development of automatic mesh simplification methods in the early 1990s, automatic LOD generation arose as an important research topic. Luebke et al. [1] provided a comprehensive introduction to the concept and history of LODs in computer graphics. In summary, the main task of an LOD is to *represent* and *generate* simpler versions of a complex model to achieve fidelity-performance tradeoffs. Usually, LODs have three types of representations: discrete LOD [10], continuous LOD [11] and view-dependent LOD [12]. De Floriani and Magillo [13] provided a survey on models and data structures of LOD representations. The mesh and shader LOD proposed in this paper can be categorized as a discrete LOD.

To generate LODs, various mesh simplification algorithms have been proposed and used. There are several surveys that have compared different techniques in theory [14], [15] and from the view of practice [16]. A popular strategy is to remove primitives greedily, leading to the lowest error. Different metrics have been proposed to guide the simplification [17]. As one of the typical methods, QEM [4] uses local geometric quadric error as the simplification metric and performs primitive delete operations. To preserve more information on the input mesh, Hoppe [5] extended it by introducing additional attributes such as color and normals. Appearance-preserving simplification [6] applies a texture deviation metric both on textures and normal maps during the simplification process. Image-driven simplification [7] introduces the error metric on final rendered images and uses it to guide the mesh simplification. In this paper, we use the image-driven error metric [7] and employ edge collapse [4] to simplify the mesh.

Shader Simplification and LOD. Recently, with the development of graphics hardware, shaders have been used as an important tool to enable complex, realistic appearance shading. The simplification of shaders and the generation of shader LODs have gradually attracted increasing attention. As pioneers, Olano et al. [2] proposed a shader simplification and LOD technique for procedural shaders. Inspired by this work, different techniques in shader simplification have been developed to different extents, such as new simplification rules [3], [8], [18], optimization strategies [19], [20], [21], [22], band-limited filtering [23], etc. Recently, He et al. [3] proposed a system that automatically generates a shader LOD. However, none of these approaches considered mesh simplification when simplifying shaders.

Aside from a shader LOD, another kind of approach, prefiltering methods, has also been developed for efficient and accurate surface shading at distant views. Texture MIP mapping [24] is one of the most widely used prefiltering techniques to create a multiscale representation of surface details. With the nonlinearity of surface geometry, normals, BRDFs, etc., different kinds of nonlinear prefiltering methods, e.g., LEAN [25]/LEADR [26] and NDF filtering [27], have been proposed and used in real-time rendering applications. More details of these methods can be found in a comprehensive survey [28]. Our paper also inherited the concept that a multiresolution representation, such as our shader and mesh LOD, should be an accurate and antialiased representation at different distances. To achieve this goal, we use supersampled, i.e., filtered, images of the original shader and mesh to guide the simplification at distant views.

Appearance-driven Joint Optimization . Differentiable rendering has recently become a trending research topic in the rendering community. By making the rendering process differentiable, scene parameters being input to the renderer can be optimized in an end-to-end way. Nvidia recently proposed a joint simplification framework that enables optimizing geometry shape and a surface appearance model encoded in different texture maps in an end-to-end fashion [29]. However, this method only considers presimplified meshes with low-resolution topology and optimizes the positions of vertices and the content of textures to match the ground truth as closely as possible. However, our proposed method focuses on automatic mesh and shader simplification, enabling the generation of object LOD.

3 Overview

Given a geometric model and a shader computing its visual appearance, our method combines two major areas of prior work: geometric simplification and shader simplification. It seeks to automatically create an LOD for the geometry and shader at the same time. In this section, we first give the problem definition, then show the error model used in simplification, and finally introduce the algorithm overview.

3.1 Problem of Mesh and Shader LOD

Let us denote the triangular mesh input as M and the input shader as S. The entire rendering process can be regarded as a function f that takes in M and S to return the color of the mesh at each pixel on the final image.

Our goal is to automatically generate a set of tuples for a simplified mesh and shader with a transition distance, (M_i, S_i, d_i), where for a camera-to-object distance range (d_i, d_{i+1}), M_i and S_i are the most preferred approximations

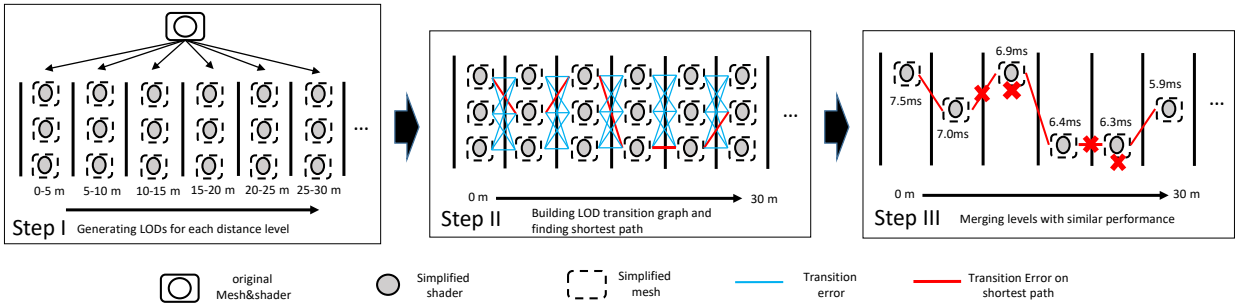


Fig. 1: Algorithm Overview.

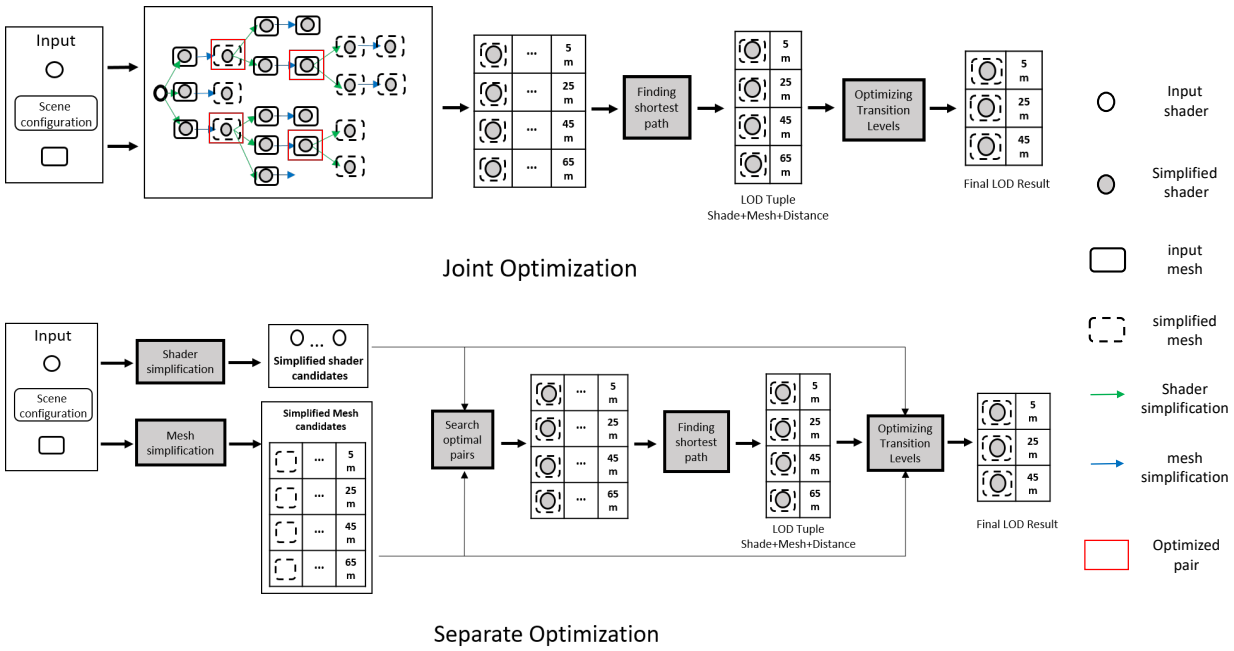


Fig. 2: Our two proposed optimization algorithms.

to M and S. Specifically, M_i and S_i should have the highest performance with acceptable output quality in that distance range. In this paper, the output quality is evaluated by two error metrics: absolute error ε_a and transition error ε_t. The absolute error ε_a captures the difference between the simplified representation (M_i, S_i) and the original representation (M, S) at distance d_i. The transition error ε_r is defined as the difference between the LOD of level i and the LOD of level $i+1$, with both rendered at distance d_{i+1}. These two error thresholds are used to ensure that these simplified versions are visually similar enough to the original version and that the transitions among them are visually smooth. Formally, these two error metrics for a tuple i, (M_i, S_i, d_i), are computed as:

$$\varepsilon_a(i) = \int_H \|f(M_i, S_i, d_i) - \overline{f}(M, S, d_i)\| dH, \tag{1}$$

$$\varepsilon_t(i) = \int_H \|f(M_i, S_i, d_{i+1}) - f(M_{i+1}, S_{i+1}, d_{i+1})\| dH, \tag{2}$$

where $\|\cdot\|$ is the norm defined on the domain H to measure the output quality, which will be specified in Section 3.2, and $\overline{f}$ is a supersampling rendering process in which a number of supersamples are created and weighted averaged at each pixel to create an accurate and anti-aliasing image of the original mesh and shader at distance d_i. Note that while the distance increases, many triangles and mapped texels of a single mesh may project to the same screen pixel. Meanwhile, if the shader produces shading with high frequency, such as specular reflections or bump-mapping surfaces, it may result in severe image aliasing at distant views [28]. To avoid such aliasing and provide multiscale surface shading in the frequency domain, we use supersampled rendering to create reference images and define errors.

Given the first quality error metric with Eq. (1), LOD generation is an optimization problem that pursues the optimal tuples, each of which (M_i, S_i, d_i) has the minimal rendering cost at the distance d_i and satisfies the quality error threshold:

$$\begin{aligned} &\arg\min_{M_i, S_i, d_i} \; t = Cost(f(M_i, S_i, d_i)), \\ &\text{s.t. } \; \varepsilon_a(i) < e_a(d_i) \cdot s_{d_i}, \end{aligned} \tag{3}$$

where t is the time cost to render (M_i, S_i) at the distance d_i, $e_a(d_i)$ is the absolute per pixel error bound at distance d_i, and s_{d_i} is the projected size of mesh M_i at distance d_i. To compute the absolute per pixel error bound at a specific

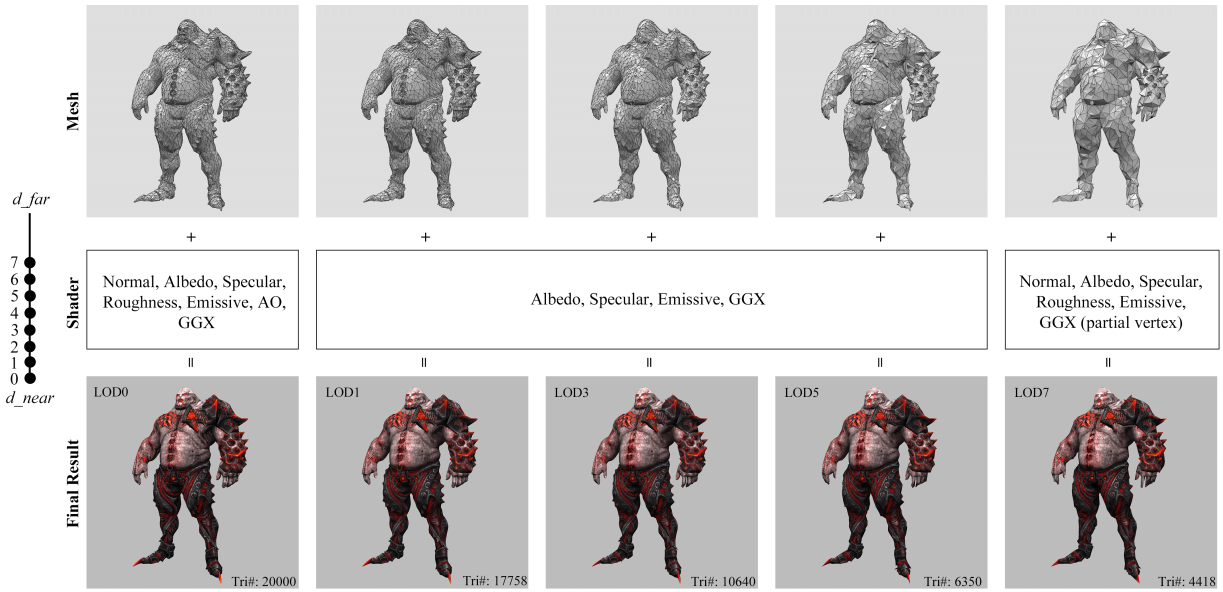


Fig. 3: One result of the proposed mesh and shader LOD generated by our method. In total, eight levels of increasingly simplified meshes and shaders are created for this Imrod model, and five of them are shown from left to right. The leftmost level is the original mesh and shader. The rightmost level is the simplest one. From top to bottom, we show the simplified mesh, simplified shader and rendered result. Our new LOD representation provides a better balance of loss of details and rendering performance, thereby achieving better performance-quality tradeoffs than prior LOD representations.

distance, we adopt the same equation that the prior work [3] used:

$$e_a(d) = (\frac{d - d_{near}}{d_{far} - d_{near}})^Q \cdot e_{max}, \tag{4}$$

where e_{max} is the maximum absolute per pixel error bound, and Q determines how rapidly quality is allowed to degrade at large viewing distances, $Q \subset [0, 1]$.

However, to keep the transitions among the generated LODs visually smooth, we first select multiple optimized pairs $\{(M, S)\}_i$ rather than just one for every level i. Then, given the transition error metric with Eq. (2), we can build a graph G that consists of paths between every combination (M_p, S_p) in $\{(M, S)_p\}_i$ of level i and every combination (M_q, S_q) in $\{(M, S)_q\}_{i+1}$ of level $i + 1$ with weight:

$$\varepsilon_t(i_p, (i+1)_q) = \int_H \|f(M_p, S_p, d_{i+1}) - f(M_q, S_q, d_{i+1})\| dH, \tag{5}$$

Then, a shortest path algorithm can be applied on G to obtain the desired LODs with the minimum total transition error.

3.2 Error Model

Our optimization depends on the quality error metric to measure differences between the simplified representation and the original representation (Eq. (1)) or between two simplified representations (Eq. (2)). In this paper, we use the *image metric*, which is a measurement of differences between pairs of images [7]. The benefit of the image metric is that all simplifications on meshes or shaders result in changes in the final images. Therefore, image errors equivalently reflect simplifications on the mesh and shader or both. Similar to Lindstrom and Turk's simplification process [7], we also render images in different view directions, V, to capture different portions of objects. In addition, the rendering function usually has a set of uniform parameters u (such as changing light positions). To capture these changes, similar to prior work [3], [8], [21], [30], we also integrate errors over the domain of uniform parameters U (including view directions and lighting configurations) to compute the final error. Formally, using the image metric, these error metrics, ε_a and ε_t, are computed as:

$$\begin{aligned}\varepsilon_a(i) &= \sum_V \int_U \iint_{xy} \|f(M_i, S_i, d_i) - \overline{f}(M, S, d_i)\|^2 dxdy, \\ \varepsilon_t(i) &= \sum_V \int_U \iint_{xy} \|f(M_i, S_i, d_{i+1}) \\ &\qquad - f(M_{i+1}, S_{i+1}, d_{i+1})\|^2 dxdy,\end{aligned} \tag{6}$$

where $\|\cdot\|^2$ is the L^2 norm of pixelwise RGB differences. In our implementation, we use *compute shader* and incremental image updates [7] to compute the image error efficiently.

3.3 Algorithm Overview

To solve the proposed problem (Section 3.1), we separate the optimization of the simplified representation and the distance and carry out the optimization of Eq. (1) and (2) in three iterative substeps. In the first substep, we divide the distance into N initial levels and optimize Eq. (3) to obtain (M_i, S_i) at each distance level. We present two algorithms, termed **interleaved optimization** and **separate optimization**, with different speed-accuracy tradeoffs to solve this optimization problem. After the optimization, multiple optimized combinations $\{(M, S)\}_i$ are selected as candidates for every level i. This optimization is independent of other levels and therefore can be solely computed at each distance level. In the second substep, we employ an adaptive optimization scheme, termed the **transition smoothing strategy**,

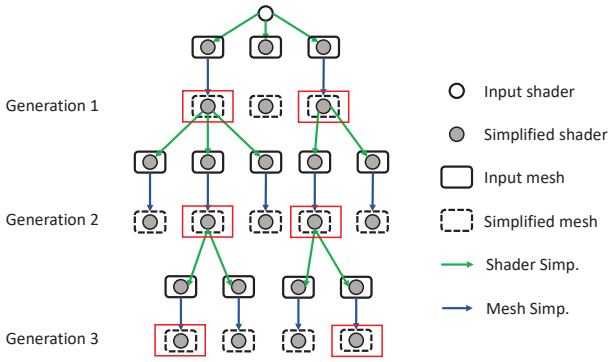


Fig. 1: An illustration of interleavingly simplifying both shader and mesh.

to create LODs. Specifically, we build a graph with weighted paths (Eq. 5) between every combination of level i and every combination of level $i+1$. Thus, the shortest path algorithm can be applied to obtain the LOD with the minimum total transition error. In the final substep, we merge distance levels if the efficiency improvement is relatively low.

The rest of the paper is organized as follows: we first introduce the two proposed algorithms, **interleaved optimization** and **separate optimization**, in Section 4 and 5, respectively. Unless otherwise noted, our discussion of the optimization strategies only focuses on a certain distance level, and it is easy to generalize to other levels. Then, we introduce the **transition smoothing** strategy used in the algorithm in Section 6. Finally, we show the experimental results validating our proposed methods in Section 7.

4 Interleaved Optimization: Spawning Optimized Pair Generations

To solve the optimization problem in Eq. (3), we propose an optimization to search for the best pair in an interleaved manner, named **interleaved optimization** in this paper. The algorithmic overview is shown at the top of Figure 2. It simultaneously explores simplified meshes and shaders to find the best simplified pairs. Technically, the input shader is first optimized with genetic programming to obtain one generation of shader variants, and then these shader variants on the Pareto frontier are individually used in an image-driven mesh simplification process to obtain pairs of simplified meshes and shaders within an error bound. All of these shader variants of those pairs with better rendering performance are selected to produce the next generation of shader variants. These steps are iteratively executed to generate a collection of optimized pairs. In this section, we first introduce the mesh and shader simplification approaches that we adopt and then present the details of the interleaved optimization algorithm.

4.1 Shader simplification

Our shader simplification follows the standard shader simplification procedure proposed in prior works [3], [8], [21]. Specifically, we first convert the shader program (including *vertex shader* and *pixel shader*) into abstract syntax trees (ASTs) and program dependence graphs (PDGs) and then apply different simplification rules to generate simplified shader variants. Three kinds of simplification rules are used in this paper. *Operation removal* [18] takes local operator reduction transformations, such as $op(a,b) \rightarrow a$ and $op(a,b) \rightarrow b$. It can simplify operations and reduce the number of instructions. *Code transformation* [8] transforms a per-pixel operation to a per-vertex or per-tessellated-vertex operation to reduce the computation in the pixel shader. *Moving to parameter* [3] defers the average substitution ($n \rightarrow average(n)$) to a new parameter stage where these averaged values are directly fed from the CPU instead of being computed at runtime in the GPU.

4.2 Mesh simplification

To simplify a mesh, we adapt the established mesh simplification framework [4] by replacing the original geometric metric with an image error metric, which is computed using the supersampled/filtered images of the original shader as the reference. In particular, we iteratively apply edge collapse operations on edges. However, the placement policy that contracts one edge into a single vertex consumes many computations, so we choose a simpler *place-to-endpoints* policy that places the vertex to the endpoint with lower image error on the edge. To preserve the topology of the simplified mesh, edges with screen-space lengths larger than D ($D = 1$ in our implementation) screen pixels would not collapse [31] during our mesh simplification.

4.3 Interleaved optimization

The interleaved simplification method directly integrates mesh simplification into shader simplification. Figure 1 shows an illustration of how the interleaved optimization works. Specifically, given an input mesh M and input shader S, we first obtain the first generation of shader variants by applying one of our shader simplification rules (Section 4.1). Then, for each newly created shader variant S_i on the Pareto frontier, we always perform progressive mesh simplification (Section 4.2) to obtain an optimally simplified mesh M_j in which the error produced by (M_j, S_i) is less than the absolute error bound at this distance. Next, those shader variants and simplified mesh pairs are sorted by their rendering performance, and the top $N\%$ ($N = 20$ in our implementation) are chosen to be the optimized pairs of this generation. By iteratively performing the above steps for every optimized pair until reaching the maximum generation depth, we can obtain generations of optimal pairs one by one. After several iterations, we obtain a collection of pairs of simplified meshes and shaders. Finally, the best pair will be selected in the transition smoothing step by considering the transition errors.

5 Separate Optimization

The interleaved optimization algorithm is nearly globally optimal but very time-consuming. Usually, it will take more than 6 hours to generate the result. To address such a limitation, we propose an approximate but more efficient algorithm, termed **separate optimization** (see the bottom of Figure 2). Specifically, the algorithm first takes a standalone simplification step to simplify the mesh and shader separately and creates sequences of increasingly simplified meshes and shaders as candidates. These pregenerated sequences of simplified meshes and shaders are organized in

order, which enables a monotonically linear search among simplified meshes and shaders, thereby reducing the 2D search problem into a 1D search and achieving a significant performance boost. In this section, we first describe how we generate the mesh and shader simplified candidates and then explain the 1D linear search algorithm that is conducted on the 2D space formed by the two sequences.

5.1 Generating Mesh Candidates

The purpose of the mesh and shader LOD is to approximate the original shading of an object at distant views. As the camera distance increases, the change in the pixel footprint enlarges the integration domain of the shading function, gradually blurring high-frequency shading and only leaving low-frequency shading. Therefore, although without any knowledge of any simplified shaders, we could use the supersampled/filtered images of the original shader at different distances to approximate the shader's simplification process and guide the mesh simplification.

While progressively simplifying meshes, we find that the error produced by some edge collapses is small and visually unnoticeable. Thus, instead of storing all progressively simplified meshes, we only select K ($K = 500$ in our implementation) simplified meshes with large error changes as candidates.

5.2 Generating Shader Candidates

By applying the simplification rules mentioned in Section 4, we are able to generate simplified shader variants with different performances and errors. However, not all simplified shader variants must be enumerated and stored. If the scene configuration is given, only those variants forming the Pareto frontier are optimal at performance or quality [21]. However, in our LOD optimization (Eq. (3)), the simplified mesh and distance in scene configuration are optimization variables and subject to change. Different simplified meshes have different numbers of vertices, and different viewing distances result in different pixels, both of which impact the performance and quality of shaders. Therefore, in this shader candidate generation step, ideally, we should select optimal shader variants for all possible scene configurations, i.e., simplified meshes and distances, and then use them in optimizing LODs.

Fortunately, we found that based on several assumptions and observations, such a shader candidate generation process can be simplified. First, as has been proven in prior work [3], the performance and error of shader variants can be predicted instead of being actually evaluated. In this way, we do not need to actually render every shader variant under all scene configurations. Second, we noted that for one shader variant with one simplified mesh, the shading errors at distances could be approximated by filtering the rendered image at the closest distance. Finally, we further observed that although these Pareto frontiers may change with scene configurations, the shader variants on Pareto frontiers are similar at similar distances and with similarly simplified meshes. This inspired us to select only representative distances and simplified meshes to compute optimal shader variants rather than exhaustively enumerating all possible scene configurations.

Algorithm 1 Adaptive LOD generation

Input: $\{\{M_k\}_i\}$,$\{S_l\}$
Output: $\{P\}$
1: // LOD distance
2: $\{d_n\} = LODDistance(N)$
3: // Pair sets of optimal simplified mesh and shader for all levels
4: $\{\{P\}_i\} = \text{ø}$
5: **for** each $d_i \in \{d_n\}$ **do**
6: $\quad\{S_l\} = SortByError(\{S_l\}, M_0, d_i)$
7: $\quad$// Search for optimal pairs of mesh and shader at d_i
8: $\quad\{P\}_i = SearchPairs(\{M_k\}_i, \{S_l\}, d_i)$
9: $\quad\{\{P\}_i\} \leftarrow \{P\}_i$
10: **end for**
11: // Find the shortest path with minumum total transition error.
12: $\{P_i\} = SmoothestPath(\{\{P\}_i\}, \{d_n\})$
13: // Merge inefficient levels
14: $\{P\} = MergeInefficientLevels(\{P_i\})$

Based on these assumptions and observations, we design the shader candidate generation steps as follows. We first uniformly choose 4 out of all N distance levels, and for each level, we select the top 10 simplified meshes generated by the previous iterative mesh simplification step, thus creating 40 combinations. Then, we adapt the genetic programming used by previous shader simplification approaches [3], [8], [21] to compute the Pareto frontiers for each combination. To efficiently perform this optimization and avoid actual evaluation, we compute the quality error from the base quality error, extend the performance model [3] to include the number of vertices and pixels, and use this model to predict the performance. Once obtaining these Pareto frontiers, we select all simplified shader variants on these frontiers as candidates. Note that the shader candidates are shared across all distance levels.

5.3 Searching Optimal Pairs of Meshes and Shaders

Given these simplified meshes and shaders produced as candidates, in this section, we introduce the algorithm to create the object LOD that selects which simplified mesh and shader are used. Shader candidates are sorted in the order of increasing errors. According to the aforementioned observation, we use the original mesh as the default mesh to sort simplified shaders. After the sorting, optimal pairs of simplified meshes and shaders are searched from the collection of simplified mesh and shader candidates, as shown in lines 6–9 of Algorithm 1.

We have a reduced optimization (Eq. (3)) that finds the optimal pair of simplified meshes and shaders with minimal rendering cost but still retains good visual fidelity (i.e., under the absolute error threshold). However, enumerating the 2D space formed by the two sequences in a brute-force manner is also time-consuming (please refer to supplemental document for more details). By utilizing the monotonicity of the quality loss of increasingly simplified meshes and shaders, we found that the search for optimal pairs of meshes and shaders can be accelerated. More specifically, we observe that by organizing mesh and shader candidates

Algorithm 2 Search Pairs

Input: $\{M_k\}$,$\{S_l\}$,d
Output: $\{P\}_{opt}$
1: $i = size(K) - 1; j = 0$
2: // All pairs with rendering costs at the convex region
3: $\{(P,t)\} = \text{ø}$
4: // Search optimal pairs along 1D boundary
5: **while** $i \geq 0$ and $j < size(L) - 1$ **do**
6: $(\varepsilon_a, t) = Evaluation(M_i, S_j, d)$
7: **if** $\varepsilon_a > e_a(d) \cdot s_d$ **then**
8: $i = i - 1$
9: **else**
10: $P_{i,j} = (M_i, S_j)$
11: $\{(P,t)\} \leftarrow (P_{i,j}, t)$
12: $j = j + 1$
13: **end if**
14: **end while**
15: $\{(P,t)\} = SortByPerformance(\{(P,t)\})$
16: // Optimal pairs
17: $\{P\}_{opt} = \text{ø}$
18: **for** each $(P,t)_n \in \{(P,t)\}$ **do**
19: // Get pairs whose rendering time is under the bound
20: **if** $(t_n - t_0)/t_0 < T$ **then**
21: $\{P\}_{opt} \leftarrow P_n$
22: **end if**
23: **end for**

in the order of increasing quality errors, those pairs satisfying the optimization constraint (Eq. (3)) form a convex region (the colored region shown in the top left of Figure 2). Assuming that the rendering time of increasingly simplified meshes is also monotonically decreasing, it is easy to prove that the optimal pair with minimal performance cost is always at the boundary of such a convex region. That is, the mesh with better performance always has a larger subscript, i.e., if $Cost(M_{k'}, S) > Cost(M_k, S)$, we have $k' < k$. Therefore, for one simplified shader, the simplified mesh with the best performance is always the mesh with an upper bound subscript, i.e., at the boundary of the region. As a result, the optimization of Eq. (3) can be converted to a 1D search along the boundary instead of a 2D search in the full space (see left bottom of Figure 2). The assumption that the rendering time of increasingly simplified meshes is also monotonically decreasing is true for most simplified meshes. The right side of Figure 2 shows a visualization of this search step from a real example, and Section 7 provides more validation of our search strategy.

We show the algorithm details in Algorithm 2. We start the search from the pair with the most simplified mesh and original shader, (M_{K-1}, S_0). Then, we iteratively minus the subscript of the simplified mesh or add the subscript of the simplified shader to visit all possible pairs along the boundary (see lines 6–15). After the search, we sort those pairs by rendering performance. If the rendering time difference between one pair P_n and the pair with the highest performance P_0 is under the bound T, we add it to the optimal pair set. Thus, we can obtain all optimal pairs of meshes and shaders at this distance.

Algorithm 3 Smoothest Path

Input: $\{\{P\}_{opt}\}$,$\{d_n\}$
Output: $\{P\}_{smt}$
1: // The LOD transition graph
2: $G = \text{ø}$
3: $i = 0$
4: **while** $i < size(N) - 1$ **do**
5: **for** each $P_a \in \{P\}_i$ **do**
6: **for** each $P_b \in \{P\}_{i+1}$ **do**
7: // Evaluate the weight by using Eq. (5)
8: $w = EvaluateTransition(P_a, P_b, d_{i+1})$
9: // Add the path from a to b with weight w to G
10: $G \leftarrow Path(a, b, w)$
11: **end for**
12: **end for**
13: **end while**
14: // Finding the smoothest path with dijkstra algorithm
15: $\{P\}_{smt} = DijkstraAlgorithm(G)$

6 Smoothing Transitions between Levels

The aforementioned optimizations are conducted locally at different levels, generating optimized pairs for each level. In this section, we propose an algorithm to finally select the LODs by minimizing transition errors and reducing transition levels.

6.1 Finding the Smoothest LODs

The pseudocode of the algorithm to select the smoothest LODs is shown in Algorithm 3. Technically, once the optimal pairs of meshes and shaders of all levels have been collected, we build an LOD transition graph by making paths from each optimal pair of one level to each optimal pair of the next level, and Eq. (5) is used to compute the weights of all paths (see lines 5–14). Then, we apply Dijkstra's algorithm [32] to find the shortest path with the minimum total transition error.

6.2 Optimizing Transition Levels

Once obtaining the optimal path, we then make a tradeoff between memory usage and rendering performance by reducing the number of levels and optimizing the distance levels. More specifically, for each level i, we first measure the rendering time $t_i = Evaluation(M_i, S_i, d_i)$, $t_{i-1} = Evaluation(M_{i-1}, S_{i-1}, d_i)$, and $t_o = Evaluation(M_0, S_0, d_i)$. If $(t_{i-1} - t_i)/t_o < W$ ($W = 4\%$ in our implementation), which means the rendering performances of these two adjacent levels are more or less the same, then level i can be deleted. After this, these inefficient levels can be eliminated, thus easing the memory usage of our LOD without losing much performance. The resulting statistics of memory savings can be found in the supplemental document.

7 Results

We implemented our method using Microsoft DirectX 11 and tested it on an NVIDIA GeForce GTX 1080 graphics card with 8 GB RAM. We evaluate our mesh and shader LOD algorithm on a collection of example objects with shaders for both deferred and forward rendering, whose statistics

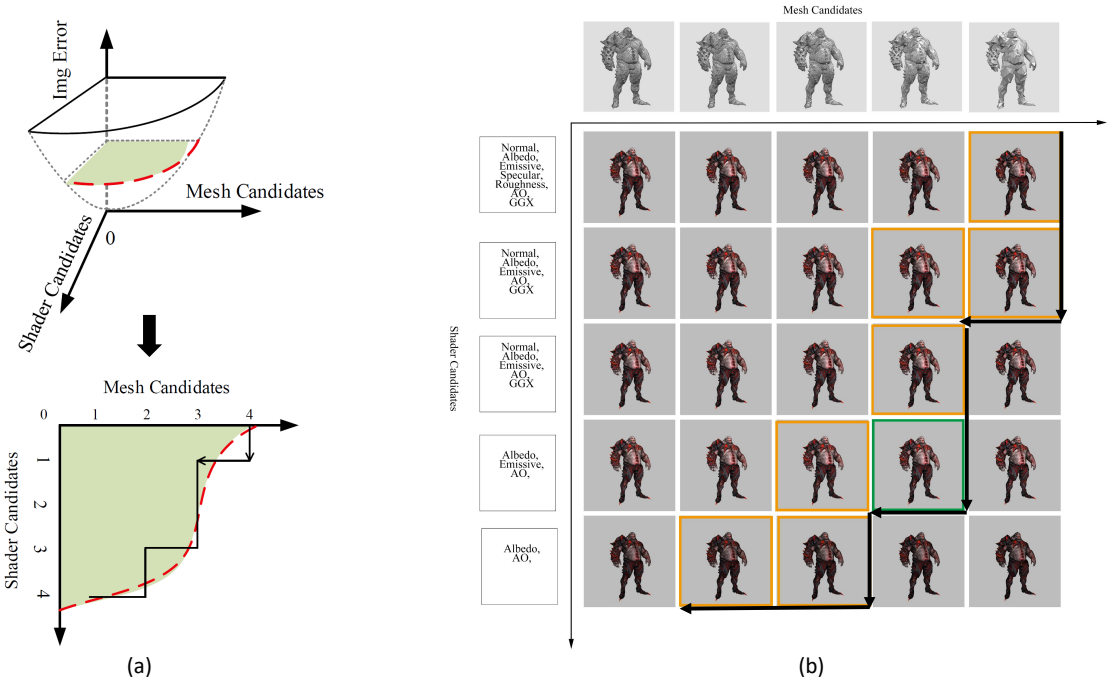


Fig. 2: (a) Illustration of the search step in 2D space to find optimal pairs of simplified meshes and shaders. (b) Visualization of linear search in Imrod Demo, where the colored boxes are the pairs along the searching path and the green box is the chosen one under the performance bound.

TABLE 1: Statistics of example objects.

	Original objects			LOD config.	
	Tris. (num)	Pixel Shader (instructions)	Memory (KB)	e_{max}	Q
Dragon	10K	5866	120	0.14	0.5
Imrod	20K	879	785	0.24	0.5
Couch	6.7K	1156	243	0.1	0.5
Rocket	3.7K	1627	161	0.1	0.6
Statue	8.9K	930	342	0.1	0.5

are listed in Table 1. During the optimization stage, we enumerate camera positions at the vertices of a Dodecahedron centered at the origin of the target object. For each camera position, we enumerate 5 light positions on a plane that is orthogonal to camera view direction with distance **d**. The 5 point light positions are located at the endpoints of a fixed cross shape. For the error computation, we enumerate all the camera-light positions and average the image space error.

In this section, we first introduce the details of the demos we test and compare the two optimizations of our method, then evaluate the separate optimization by analyzing the generated LODs and comparing with prior work. We Finally evaluate our method on game assets with more complex materials and geometries. Please refer to the supplementary materials for more evaluation and visulization results.

The Imrod is modified from the prior work of Wang et al. [8]. The shader computes shading from a physically based shading model using GGX [33] with multiple material textures. The Dragon is adopted from the prior work of Sitthi et al. [20]. The shader includes a texture function with four octaves (generated with a standard procedural 3D noise function) and a Phong BRDF model. These two objects are rendered in a forward rendering framework.

The Couch, Rocket and Statue are obtained from the Unreal Marketplace [34]. Both shaders perform pattern generation to emit a G-buffer for subsequent standard lighting. Our method only simplifies the pattern generation process so that it can be naturally integrated in a deferred rendering pipeline. The error of simplified meshes and shaders under deferred rendering scenarios is still evaluated by pixelwise differences of final shading colors.

The maximum absolute per pixel error e_{max} and the quality factor Q are set per object and reported in Table 1. To initialize the adaptive LOD generation, we set 10 levels as defaults.

7.1 Comparison of Two Optimizations

We compare the performance of the LODs generated with our two proposed optimization algorithms. As seen in columns 5 and 6 of Table 2, the rendering performances of the results produced by these two optimization strategies are very close to each other. The time consumed by interleaved optimization heavily depends on the complexity of the input mesh and the input shader. For example, the Imrod demo has 20K triangles and performs interleaved optimization for approximately 15 hours (see column 11) to simplify meshes for approximately 120 optimized shader variants at all 10 levels. In comparison, the Rocket demo with 3.7K triangles requires approximately 9 hours of interleaved optimization to simplify meshes for 210 optimized shader variants at all 10 levels. However, with separate optimization, for each level, the input mesh is simplified once, and the combined mesh-shader pairs are evaluated several times, requiring only tens of minutes.

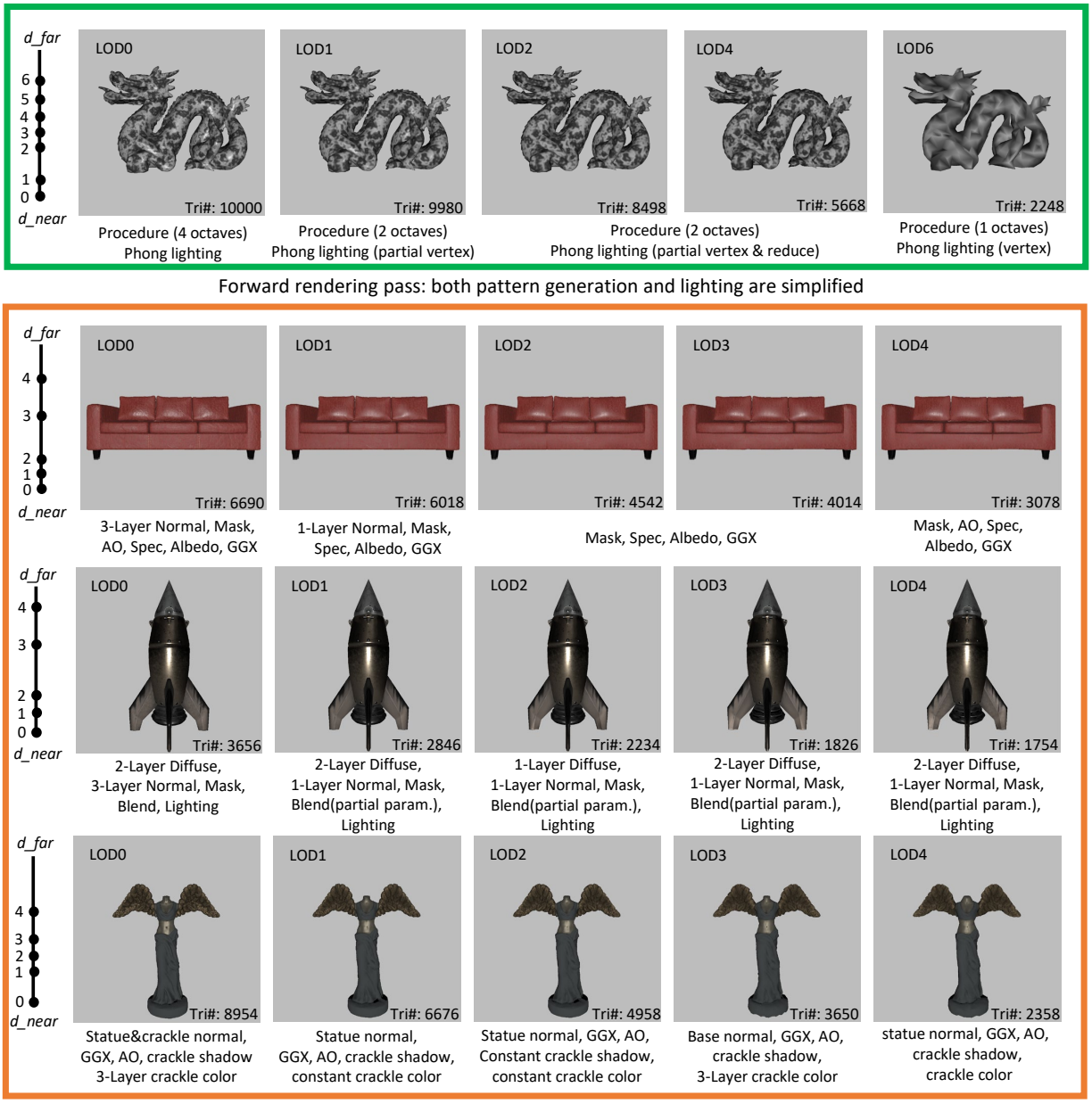


Fig. 3: Results of mesh and shader LODs generated by separate optimization.

7.2 Evaluation of Separate Optimization

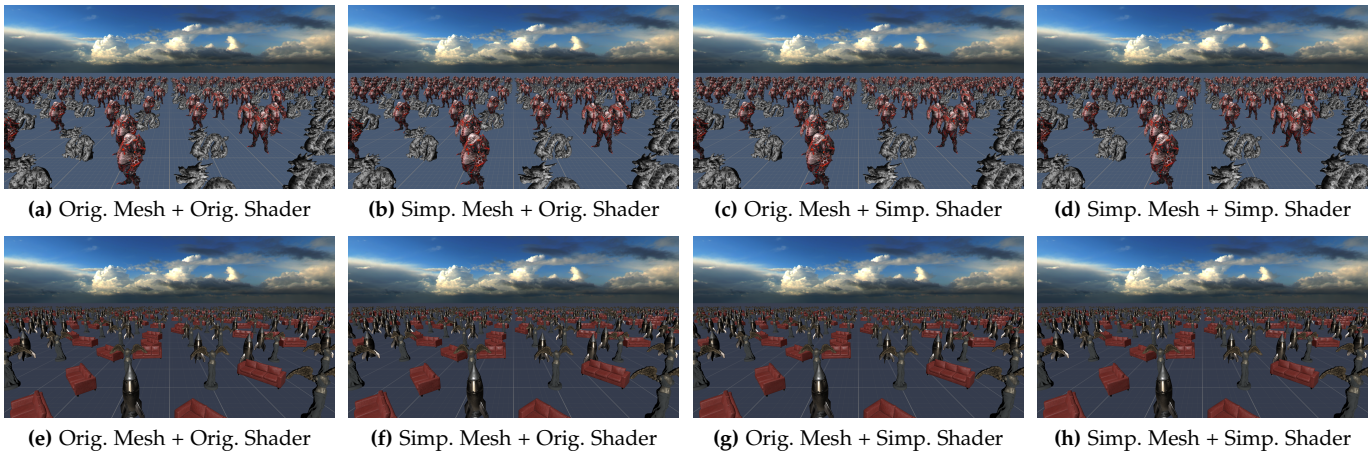
The comparison of two optimizations shows that separate optimization gives similar quality results but is significantly faster. Therefore, in the results section, we focus more on separate optimization and provide an extra analysis of the results generated by it. We first present the examples of the generated LODs, including all of the demos mentioned above, and then analyze the performance of the separate optimization, compared with prior works [3] [7].

7.2.1 Generated LODs

In Figure 3, we show five levels of the Imrod LOD. The three rows from top to bottom present the simplified meshes, the simplified shading functions, and the final shading results. The viewing distances for using different shader and mesh pairs in the policy are plotted as dots on the left.

More results of other example objects are presented in Figure 3. For each example object, we select 5 LODs to present in the figure. From left to right, the results are visualized from the low-distance level to the high-distance level. In each image, we use $Tri\#$ to denote the number of triangles in the simplified mesh and briefly describe the simplified shader under each image. For example, "Normal, Albedo, Specular" indicates that this shader uses multiple textures in shading, including normal maps, albedo maps, and specular maps.

"Phong lighting(vertex)" indicates that the shading computations of Phong lighting are moved from pixel shader to vertex shader and it is only performed per vertex. "Phong lighting(partial reduce)" indicates that some shading computations of Phong lighting are reduced. "Blend(partial



(a) Orig. Mesh + Orig. Shader **(b)** Simp. Mesh + Orig. Shader **(c)** Orig. Mesh + Simp. Shader **(d)** Simp. Mesh + Simp. Shader

(e) Orig. Mesh + Orig. Shader **(f)** Simp. Mesh + Orig. Shader **(g)** Orig. Mesh + Simp. Shader **(h)** Simp. Mesh + Simp. Shader

Fig. 4: Screen shots of different LODs on two test scenes: no LOD (original mesh and shader), mesh LOD (simplified mesh with original shader) [7], shader LOD (original mesh with simplified shaders) [3], and our mesh and shader LOD.

TABLE 2: Left: performance comparison of four LODs (Figure. 4). In the bracket, we show the speedup factor compared with the end-to-end rendering time using no LOD. Right: time statistics of LOD generation of example objects.

	Rendering Time (ms)					LOD generation Time with 1D search(s)				2D Search	
	No LOD	Mesh LOD	Shader LOD	Sep. LOD	Inter. LOD	Mesh	Shader	Optimization	Total	Total Time (s)	Inter Opt.(s)
Dragon	5.03 (1.0×)	3.23 (1.56×)	3.35 (1.50×)	1.54 (3.27×)	**1.52 (3.30×)**	598	206	46	850	6592	44720
Imrod	3.75 (1.0×)	1.52 (2.47×)	2.66 (1.41×)	1.03 (3.64×)	**1.00 (3.75×)**	2241	418	160	2819	23118	56478
Couch	1.50 (1.0×)	1.00 (1.50×)	1.19 (1.26×)	**0.73 (2.08×)**	0.74 (2.03×)	301	349	65	715	9122	36917
Rocket	0.77 (1.0×)	0.58 (1.33×)	0.68 (1.13×)	0.53 (1.45×)	**0.47 (1.64×)**	124	613	59	796	9861	31693
Statue	1.52 (1.0×)	0.66 (2.30×)	1.20 (1.26×)	0.48 (3.16×)	**0.44 (3.45×)**	931	498	107	1536	13254	37548
Mix Dragon, Imrod	4.24 (1.0×)	2.11 (2.00×)	2.90 (1.46×)	1.26 (3.37×)	**1.25 (3.39×)**	-	-	-	-	-	-
Mix Couch, Rocket, Statue	1.20 (1.0×)	0.68 (1.76×)	1.06 (1.13×)	0.58 (2.07×)	**0.54 (2.22×)**	-	-	-	-	-	-

param.)" indicates that when performing blending operations in the pixel shader, some parameters are omitted so that some part of the computation is simplified. Please refer to the supplementary video for more visualization of different levels.

From these results, it can be seen that with increasing distance, our LODs balance the geometry and shading details well and produce high-quality tradeoffs. Most importantly, by taking both the geometric and shading details into consideration, our method enables new possibilities of simplifications.

The Couch demo shows a good example. From level 3 (LOD3) to level 4 (LOD4), our LOD decreases the geometric details (from 4.0k triangles to 3.1k triangles) but increases shading details by bringing back the emitted ambient occlusion map at level 0 (LOD0). A similar tradeoff can also be found in the Imrod demo and the Rocket demo. Such results imply that as the camera-to-object distance increases or the pixel footprints covered by the object decrease, our simplification algorithm tends to perform more simplification on the mesh rather than the shader.

7.2.2 Comparison with Prior Work

Compared with prior works that only simplify meshes [7] or shaders [3], our methods extend the tradeoffs between the performance and the rendering quality, thereby achieving better performance at the same quality. To demonstrate such an advantage, we compare three configurations: an original mesh and shader without an LOD, a mesh LOD only [7], and a shader LOD only [3]. The latter two LODs are both created using the same error thresholds as ours. We generate six test scenes, including four scenes with one example object and two scenes with two objects. In these scenes, approximately 400 objects on the ground are captured by a walk-through camera. The time statistics of these comparisons are shown in Table 2, and screenshots of the two test scenes are shown in Figure 4. All walk-through sequences can be found in the supplementary video.

Compared with the configuration without an LOD, our separate optimization could achieve 1.45× to 3.64× speedup but still retain high visual quality. Even with a side-by-side comparison (in the supplementary video), the visual difference is unnoticeable. This shows that the generated LODs well preserve the constraints both at the absolute error and the transition error.

Compared with the mesh LOD [7], our separate optimization could achieve a 1.09× to 2.10× speedup. Such benefits mainly come from the complexity reduction of the shader code. The performance gain of the Dragon demo with the most complex shader is most significant, with a 52% GPU time savings. However, for objects with simple fragment shading, such as Rocket, the performance gain reduces to 9%. Overall, our LODs enable more possibilities of simplified representations of objects and achieve better performance.

Thanks to the reduced complexities of meshes, our separate optimization could achieve 1.33× to 2.58× speedup compared with the shader LOD [3]. The Imrod demo, having the most triangles among the four example objects, exhibits the most significant speedup with our method.

7.2.3 Test on Complex Game Assets

To further test the suitability of the method for game assets involving more complex meshes and materials, we adopt the Trooper model from the Matinee Demo, which is also found in the Unreal Marketplace [35] and shown in Figure 5. This model consists of 5 sub-meshes with a total of 42.6k triangles using 6 different materials. Each sub-mesh uses 4-5

TABLE 3: Detailed statistics of the Trooper model.

Sub-meshes	Original objects			LOD config.	
	Tris. (num)	Pixel Shader (instructions)	Memory (KB)	e_{max}	Q
Body	5.8K	2462	540	0.2	0.5
Arm	13.3K	2541	1227	0.2	0.5
Leg	4.5K	2746	1087	0.2	0.5
Accessory	10.8K	2076	400	0.2	0.5
Mask	8.3K	1830	713	0.2	0.5
Total	42.6K	10538	14505	-	-

TABLE 4: Performance of the Trooper model.

	No LOD	Shader LOD	Mesh LOD	Sep. LOD
Rendering Time	3.00ms(1.0x)	2.72ms(1.10x)	1.77ms(1.69x)	**1.55ms(1.93x)**

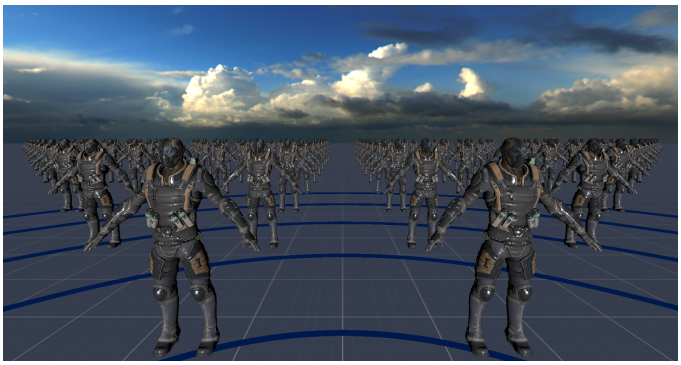


Fig. 5: The Trooper model.

materials with different parameters and blending strategies. The detailed statistics of the Trooper model are provided in Table 3. We conduct our separate optimization on each sub-mesh and render them all together, while different sub-meshes are represented via different LODs. Table 4 shows the rendering performance on an NVIDIA GTX1080.

8 Discussion and Future Work

This paper presents a new discrete multiresolution representation of objects, which includes the mesh and shader LOD. It is based on the observation that the simplification of appearance could occur in both at geometric and shading details. Therefore, at each level of detail, a simplified mesh with a simplified shader provides better balances of the loss of details and the rendering performance. The results show that our mesh and shader LOD achieves better performance-quality tradeoffs than other LOD representations, such as only using simplified meshes or shaders.

We present two optimization algorithms to create this new LOD representation. The interleaved optimization of meshes and shaders adopts genetic programming to compute the Pareto frontiers for each combination, thereby providing the best quality. To achieve a better balance between performance and quality, we further propose separate optimization utilizing the monotonicity of quality loss of increasingly simplified meshes and shaders to reduce the optimization search from a 2D space to a 1D curve. The separate manner significantly reduces computational time but obtains similar quality.

Several limitations still exist with our method. Meshes and shaders need to be complex enough to provide a large space for our algorithm to explore. Additionally, our algorithm performs best on assets with balanced meshes and shaders. If a large mesh is combined with a simple shader, our method degenerates to mesh LOD generation and vice versa. Also, the LOD generation of our method is a precomputation process, and the lighting situation we try to enumerate is limited and cannot cover all the cases in real games. If the runtime lighting deviates largely from the training process, our generated LODs will perform worse. On the other hand, our optimization process considers only a single object, limiting the use of shaders with complex GI effects such as screen space reflection (SSR). However, these are common limitations for all shader LOD generation methods with precomputation processes, which can be alleviated by fine tuning the precomputation process to consider the runtime light transport situations of specific scenes.

Our work is the first LOD representation considering both geometry and shaders. There are several potential directions to explore this idea further. First, in this paper, we only propose a discrete LOD representation. This is good at simplifying single objects but is unable to handle other objects, such as terrain, that require a continuous LOD. It would be an interesting topic to explore a continuous mesh and shader LOD in the future. Additionally, in this work, we assume that the object has only one shader. However, in practice, one object may have multiple shaders. Therefore, it would be another interesting future topic to consider more constraints of visual details generated by multiple shaders on one object and design new mesh and shader simplification algorithms. Finally, our fully automatic simplification provides great convenience for users. However, it would also be interesting to let users take part in the optimization process, for example, by letting the user manually edit some simplified mesh and then put the mesh into the optimization or let the user set some features as constraints in the optimization.

Acknowledgments

The authors would like to thank all reviewers and editors for their thoughtful comments. 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.

References

[1] D. P. Luebke, *Level of detail for 3D graphics*. Morgan Kaufmann, 2003.

[2] M. Olano, B. Kuehne, and M. Simmons, "Automatic shader level of detail," in *Proceedings of Graphics Hardware*, 2003, pp. 7–14.

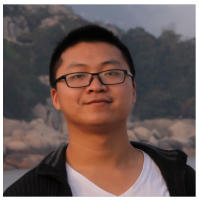
[3] Y. He, T. Foley, N. Tatarchuk, and K. Fatahalian, "A system for rapid, automatic shader level-of-detail," *ACM Trans. on Graph. (TOG)*, vol. 34, no. 6, p. 187, 2015.

[4] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216.

[5] H. Hoppe, "New quadric metric for simplifying meshes with appearance attributes," in *Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)*, ser. VISUALIZATION '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. –.

[6] J. Cohen, M. Olano, and D. Manocha, "Appearance-preserving simplification," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 1998, pp. 115–122.
[7] P. Lindstrom and G. Turk, "Image-driven simplification," *ACM Transactions on Graphics (ToG)*, vol. 19, no. 3, pp. 204–241, 2000.
[8] R. Wang, X. Yang, Y. Yuan, W. Chen, K. Bala, and H. Bao, "Automatic shader simplification using surface signal approximation," *ACM Trans. on Graph. (TOG)*, vol. 33, no. 6, p. 226, 2014.
[9] P. Sitthi-Amorn, N. Modly, W. Weimer, and J. Lawrence, "Genetic programming for shader simplification," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 1–12, 2011.
[10] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," *Communications of the ACM*, vol. 19, no. 10, pp. 547–554, 1976.
[11] H. Hoppe, "Progressive meshes," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/237170.237216
[12] ——, "Smooth view-dependent level-of-detail control and its application to terrain rendering," in *Proceedings Visualization '98 (Cat. No.98CB36276)*, 1998, pp. 35–42.
[13] L. De Floriani and P. Magillo, "Multiresolution mesh representation: Models and data structures," in *Tutorials on Multiresolution in Geometric Modelling*. Springer, 2002, pp. 363–417.
[14] P. Cignoni, C. Montani, and R. Scopigno, "A comparison of mesh simplification algorithms," *Computers & Graphics*, vol. 22, pp. 37–54, 1997.
[15] P. S. Heckbert and M. Garland, "Survey of polygonal surface simplification algorithms," SIGGRAPH 1997 Course #25 Notes, 1997.
[16] D. P. Luebke, "A developer's survey of polygonal simplification algorithms," *IEEE Comput. Graph. Appl.*, vol. 21, no. 3, pp. 24–35, May 2001.
[17] O. Matias van Kaick and H. Pedrini, "A comparative evaluation of metrics for fast mesh simplification," in *Computer Graphics Forum*, vol. 25, no. 2. Wiley Online Library, 2006, pp. 197–210.
[18] F. Pellacini, "User-configurable automatic shader simplification," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 445–452, 2005.
[19] D. Scherzer, S. Jeschke, and M. Wimmer, "Pixel-correct shadow maps with temporal reprojection and shadow test confidence," in *Proceedings of the 18th Eurographics conference on Rendering Techniques*. Eurographics Association, 2007, pp. 45–50.
[20] P. Sitthi-amorn, J. Lawrence, L. Yang, P. V. Sander, D. Nehab, and J. Xi, "Automated reprojection-based pixel shader optimization," in *ACM SIGGRAPH Asia 2008 papers*, 2008, pp. 1–11.
[21] P. Sitthi-Amorn, N. Modly, W. Weimer, and J. Lawrence, "Genetic programming for shader simplification," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6. ACM, 2011, p. 152.
[22] Y. He, T. Foley, and K. Fatahalian, "A system for rapid exploration of shader optimization choices," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 112, 2016.
[23] J. Dorn, C. Barnes, J. Lawrence, and W. Weimer, "Towards automatic band-limited procedural shaders," in *Computer Graphics Forum*, vol. 34, no. 7. Wiley Online Library, 2015, pp. 77–87.
[24] L. Williams, "Pyramidal parametrics," *SIGGRAPH Comput. Graph.*, vol. 17, no. 3, p. 1–11, Jul. 1983. [Online]. Available: https://doi.org/10.1145/964967.801126
[25] M. Olano and D. Baker, "Lean mapping," in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2010, pp. 181–188.
[26] J. Dupuy, E. Heitz, J.-C. Iehl, P. Poulin, F. Neyret, and V. Ostromoukhov, "Linear efficient antialiased displacement and reflectance mapping," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 211:1–211:11, 2013.
[27] C. Han, B. Sun, R. Ramamoorthi, and E. Grinspun, "Frequency domain normal map filtering," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 28:1–28:12, 2007.
[28] E. Bruneton and F. Neyret, "A survey of nonlinear prefiltering methods for efficient and accurate surface shading," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 2, pp. 242–260, 2012.
[29] H. Jon, M. Jacob, L. Jaakko, M. Aittala, and S. Laine, "Appearance-driven automatic 3d model simplification," in *Computer Graphics Forum*. Wiley Online Library, 2021.
[30] F. Pellacini, K. Vidimče, A. Lefohn, A. Mohr, M. Leone, and J. Warren, "Lpics: a hybrid hardware-accelerated relighting engine for computer cinematography," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 464–470, 2005.
[31] "simplygon," [EB/OL], http://www.https://www.simplygon.com/ Accessed July 7, 2020.
[32] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, p. 269–271, Dec. 1959. [Online]. Available: https://doi.org/10.1007/BF01386390
[33] S. McAuley, S. Hill, A. Martinez, R. Villemin, M. Pettineo, D. Lazarov, D. Neubelt, B. Karis, C. Hery, N. Hoffman, and H. Zap Andersson, "Physically based shading in theory and practice," in *ACM SIGGRAPH 2013 Courses*, ser. SIGGRAPH '13, 2013, pp. 22:1–22:8.
[34] UnrealEngine, "Unreal engine 4 documentation," *https://docs.unrealengine.com/latest/INT/index.html*, 2016.
[35] EpicGames, "Matinee demo," https://www.unrealengine.com/marketplace/en-US/learn/matinee.

Yuzhi Liang is now a Ph.D. candidate in the State Key Laboratory of CAD&CG, Zhejiang University. His interests are in performance optimization in real-time rendering, including multi-resolution representation and adaptive rendering.



Qi Song received a bachelor's degree in Software Engineering in 2015 and a master's degree in Computer Science in 2018, both from Zhejiang University. He is now a developer in Booming Tech. His interests are in multi-resolution representation and game engine architecture.

Rui Wang is a professor at the State Key Laboratory of CAD&CG, Zhejiang University. He received a bachelor's degree in Computer Science and a Ph.D. degree in Mathematics from Zhejiang University. His research interests are mainly in real-time rendering, realistic endering, GPU-based computation, and 3D display techniques. Now, he is leading a group working on the next-generation real-time rendering engine and techniques

Yuchi Huo is a "Hundred Talent Program" researcher in State Key Lab of CAD&CG, Zhejiang University. His research interests are in rendering, deep learning, image processing, and computational optics, which are aiming for the realization of next-generation neural rendering pipeline and physical-neural computation.

Hujun Bao is a professor with the State Key Laboratory of CAD&CG and the College of Computer Science and Technology, Zhejiang University. He leads the 3D graphics computing group in the lab, which mainly makes researches on geometry computing, 3D visual computing, real-time rendering, and their applications. His research goal is to investigate thefundamental theories and algorithms to achieve good visual perception for interactive digital environments, and develop related systems.