



**HAL**  
open science

# A View-Dependent Metric for Patch-Based LOD Generation & Selection

Thibaud Lambert, Pierre Bénard, Gael Guennebaud

► **To cite this version:**

Thibaud Lambert, Pierre Bénard, Gael Guennebaud. A View-Dependent Metric for Patch-Based LOD Generation & Selection. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2018, 1 (1), 10.1145/3203195 . hal-01779816

**HAL Id: hal-01779816**

**<https://inria.hal.science/hal-01779816>**

Submitted on 27 Apr 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# A View-Dependent Metric for Patch-Based LOD Generation & Selection

THIBAUD LAMBERT, Inria Bordeaux Sud-Ouest, France

PIERRE BÉNARD, LaBRI (UMR 5800, CNRS, Univ. Bordeaux), France

GAËL GUENNEBAUD, Inria Bordeaux Sud-Ouest, France

With hardware tessellation, highly detailed geometric models are decomposed into patches whose tessellation factor can be specified dynamically and independently at render time to control polygon resolution. Yet, to achieve maximum efficiency, an appropriate factor needs to be selected for each patch according to its content (geometry and appearance) and the current viewpoint distance and orientation. We propose a novel patch-based error metric that addresses this problem. It summarizes both the geometrical error and the texture parametrization deviation of a simplified patch compared to the corresponding detailed surface. This metric is compact and can be efficiently evaluated on the GPU along any view direction. Furthermore, based on this metric, we devise an easy-to-implement refitting optimization that further reduces the simplification error of any decimation algorithm, and propose a new placement strategy and cost function for edge-collapses to reach the best quality/performance trade-off.

CCS Concepts: • **Computing methodologies** → **Rendering**; *Mesh geometry models*;

Additional Key Words and Phrases: level-of-details, mesh simplification, hardware tessellation

## ACM Reference Format:

Thibaud Lambert, Pierre Bénard, and Gaël Guennebaud. 2018. A View-Dependent Metric for Patch-Based LOD Generation & Selection. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1 (January 2018), 21 pages. <https://doi.org/10.1145/3203195>

## 1 INTRODUCTION

To achieve real-time performances, extremely detailed geometric models are usually rendered using Level-of-Details (LOD) [Luebke et al. 2002]. LOD are *generated* during a pre-process: the detailed mesh is iteratively simplified and stored in a compact hierarchical representation. At render time, the appropriate level in the hierarchy is *selected* according to a view-dependent criterion. If this criterion is properly chosen, the surface complexity is significantly reduced while its visual appearance is preserved. The visual appearance of a 3D model depends on numerous parameters: not only its geometry, but also its material properties, the illumination environment and the camera viewpoint. For real-time applications, no assumption on the illumination environment nor the surface material can be made, because those can change depending on the 3D scene, the different instances of the same model in the scene, or even through time. Consequently most approaches focus on measuring the geometrical error, and sometimes the deviation of surface attributes (e.g., colors, normals), according to the viewing distance and direction.

Among them, a well-known technique is the Progressive Meshes [Hoppe 1997] and their parallel GPU implementations [Derzaf and Guthe 2012; Liang Hu et al. 2010]. This is a fine-grained approach working at the level of individual edges, and most efforts in terms of LOD generation

---

Authors' addresses: Thibaud Lambert, Inria Bordeaux Sud-Ouest, France; Pierre Bénard, LaBRI (UMR 5800, CNRS, Univ. Bordeaux), France; Gaël Guennebaud, Inria Bordeaux Sud-Ouest, France.

---

© 2018 Association for Computing Machinery.

This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, <https://doi.org/10.1145/3203195>.

and selection have been conducted in this context. However, on contemporary GPUs, much higher geometry throughput is achieved using the dedicated tessellation unit, which enables a dynamic control of the mesh resolution on a per patch basis. For a comprehensive survey of real-time rendering techniques with hardware tessellation please see the state-of-the-art report of Nießner et al. [2016]. The specificities of the hardware tessellator imply new challenges for both the generation and selection of LOD: an appropriate level must be selected for an *entire* patch, and each generated level needs to match the *predefined, uniform* subdivision pattern.

The LOD generation problem has been partly solved by the method of Lambert et al. [2016]. Storing displacements and surface attributes on a per vertex basis [Schäfer et al. 2013], they showed how classical feature-adaptive decimation algorithms based on edge-collapses can be adapted to generate LOD compatible with hardware tessellation. More accurate simplification rules have yet to be defined though.

During rendering, each patch can be subdivided at a different tessellation factor. The best approaches are currently limited to isotropic metrics, such as the Hausdorff distance between each LOD and the detailed mesh [Lambert et al. 2016; Schäfer et al. 2013], which overestimates the error along some view directions and does not take into account the surface attributes.

So far the generation and selection of LOD have been mostly considered as two orthogonal problems, whereas, as a matter of fact, both are highly interconnected. Indeed, the challenge of selection is to quantify with maximal accuracy the error introduced by a given LOD for a given viewing distance and direction. Assuming we have at our disposal such an error metric, then the role of LOD generation is to minimize the exact same error, but for all view directions in a pre-process.

*Contributions.* Based on this observation, our first and central contribution is a patch-based view-dependent metric which estimates both the geometric and attributes distance between the LOD and the reference detailed mesh. Our approach draws its inspiration from Cohen et al. [1998] attributes-preserving mesh simplification method. Similarly to this work, we assume that the attributes, such as texture colors and normals, are decoupled from the geometry and stored into 2D textures. Measuring the attribute deviation then boils down to measuring the *texture parameterization* deviation. It is accomplished by considering *difference vectors* matching points with the same texture coordinates on the LOD and reference surface. We show that the square error induced by these vectors can be tightly and conservatively summarized by a symmetric quadratic polynomial, which enables a fast GPU evaluation of the approximation error along any view direction.

This new error metric can not only be used to compare different LOD generation techniques, but even drive the generation process. Since directly minimizing such an error metric would be intractable, we designed three separate components whose combination allows to reduce the error metric by up to a factor 2.5. The first one aims at minimizing the magnitude of the previous difference vectors through a global refitting step of the vertex positions while assuming fixed texture coordinates. This optimization can be applied on top of any decimation algorithm, as long as the texture coordinates are well estimated after each edge-collapse. Second, observing that this refitting step also ensures volume preservation, we devise a novel edge-collapse simplification heuristic extending Lindstrom and Turk [1998] volume optimization constraint to compute simultaneously the optimal position and texture coordinates of the resulting vertex in a scale invariant fashion. Finally, we define a new cost for edge-collapse simplification which favors edges that do not increase our novel error metric. In particular, we show how the error variation that would be introduced by collapsing each edge can be quickly estimated.

We show that those enhancements permit to reduce the number of tessellated triangles by a factor of four for the same visual quality, leading to substantial speedups ( $\times 3$  on average).

## 2 RELATED WORK

We first provide a brief overview of the different methods proposed for LOD generation, and then review approaches to measure attributes deviation. Finally, we discuss the LOD selection metrics that are used in the context of hardware tessellation.

*LOD Generation.* Most simplification algorithms are based on a successions of *edge-collapse* operations driven by (1) a placement procedure to determine the position of the merged vertex, and (2) a local error estimation used to order the edges. These two components can be based on the same ingredient, as in the famous Quadratic Error Metric (QEM) [Garland and Heckbert 1997], or be partly separated as in the Lindstrom and Turk's metric (LTM) [1998]. We refer to the report of Cignoni et al. [1998a] for a survey and comparison of the numerous variants that have been proposed. These approaches share the same limitation: they are based on a very local estimation of the error, which does not give any indication about the global approximation error made at the end of the process. Some approaches thus directly measure the geometric deviation during the simplification by computing an exact or approximated Hausdorff distance [Ciampalini et al. 1997; Cohen et al. 1996; Klein et al. 1996].

When working with hardware tessellation engines, classic implementations simply store displacements and surface attributes into mip-mapped textures. Some methods strive to optimize individual levels [Jang and Han 2013; Yuan et al. 2016], but cannot produce continuous LOD. Higher accuracy is achieved by storing them on a per vertex basis via an adequate multi-resolution data structure [Schäfer et al. 2013]. Extending this storage scheme, Lambert et al. [2016] showed how feature-adaptive decimation algorithms based on edge-collapses can be adapted to *strip-collapses* to match the fixed GPU tessellation pattern.

*Attributes deviation metrics.* Distortions or shift of the attributes can have a major impact on the visual appearance of the mesh. Several perceptually-inspired metrics have been designed to compare meshes in the context of watermarking, compression and deformations [Corsini et al. 2013]. Such methods either need to make several hypothesis on the lighting and display conditions, or can hardly handle the impact of LOD transitions at runtime. In the remaining of this paper, we thus focus on objective metrics.

In the context of progressive meshes, the QEM has been extended to take into account mesh attributes [Garland and Heckbert 1998; Hoppe 1999]. However these metrics are very local and cannot summarize the error over an entire patch. Roy et al. [2004] proposed a more general estimate of the attributes deviation by considering the difference between the attributes of a point on a given surface and the attributes of the nearest point on the reference surface. Other metrics focus on measuring the differences between the attributes of two meshes. However, for all these metrics, the 3D mesh needs to be scaled such that the attributes and the positions have similar magnitude.

Cohen et al. [1998] show that the geometrical and attribute errors can be advantageously combined by storing attributes (e.g., texture colors, normals) within 2D textures, and then considering the distance between a given point on the first surface and the point with the same texture coordinates on the other surface. Williams et al. [2003] extend this approach to generate and navigate into continuous LOD with a perceptual model. Sanders et al. [2001] use Cohen's metric to generate texture atlases for progressive meshes. Alternatively, Willmot [2011] extends vertex clustering simplification [Rossignac and Borrel 1993] to handle vertex attributes. Although this method is appealing for its robustness and efficiency, it does not match the ability of edge-collapse approaches to preserve fine features.

*LOD selection with hardware tessellation.* The simplest selection scheme consists in computing the patch tessellation factor based on the minimum distance between the midpoint of the patch

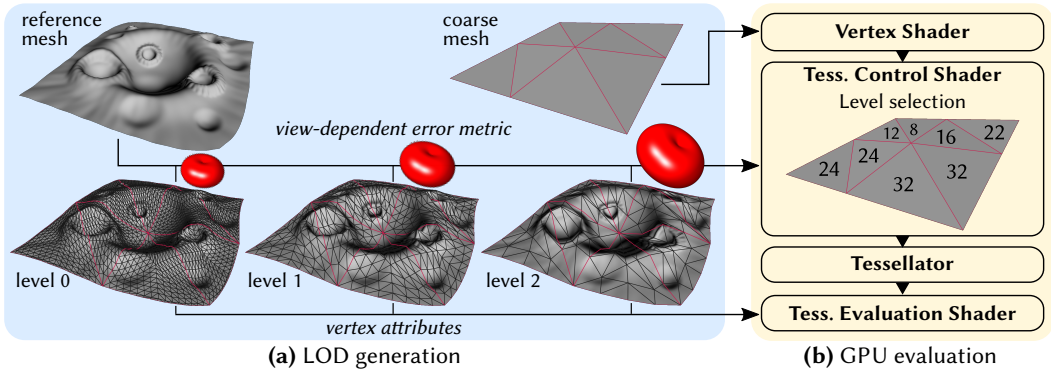


Fig. 1. Full processing pipeline — (a) as a pre-process, the LOD is generated from the reference mesh by decimation and, for each patch, at each level, its approximation error with respect to the reference surface is summarized into a compact view-dependent metric, (b) these are then used during hardware tessellation to select the most appropriate patch level according to the current viewing distance and direction.

edges and the viewpoint. This approximation is very crude since it does not take into account the length of the edge. To do so, Cantlay [2001] proposes to fit a sphere around each edge of the patch, and to choose the tessellation factor based on the maximum projected diameter in screen space. This method is still very fast and easy to compute, but it does not take into account the actual patch content after tessellation (i.e., its displaced geometry and appearance).

The Hausdorff distance has been widely used to evaluate and compare the result of simplification algorithms [Cignoni et al. 1998b; Silva et al. 2009]. A natural approach thus consists in pre-computing and storing for each patch the Hausdorff distance between each LOD and the detailed mesh [Schäfer et al. 2013]. During rendering, the LOD is selected such that the Hausdorff distance is strictly inferior to a user-defined pixel bound. Although this method gives an upper bound of the introduced geometric error, it may overestimate the error along some view directions, and it does not take into account the attributes of the surface. For example, consider a given LOD of a height-field: the geometrical error is much lower when seen from above than from a grazing viewpoint, because most of the approximation is made along the direction of elevation. On the contrary, if this height-field is textured, the parametrization distortions are likely most visible from above.

### 3 OVERVIEW

Our method takes as input a detailed reference mesh and a corresponding decomposition into fully tessellated patches, called *level 0*, matching the fixed hardware tessellation pattern (see Figure 1). In the spirit of [Cohen et al. 1998], we require that these two meshes are consistently parameterized. Depending on the production pipeline, the level 0 can be obtained by different means; one option is to start from a coarse decomposition into patches uniformly tessellated in the parametric space, and to sample the reference mesh or a displacement map, as detailed in Section 6.1. As in most modern rendering pipelines, we further assume that all spatially varying attributes, including normals and texture colors, are stored within texture maps. This way, attribute distortions are conservatively measured by the parametric distortions. From those two input meshes, our method produces (1) a hierarchy of LOD compatible with hardware tessellation, and (2) a view-dependent metric measuring, for each patch and each level of the hierarchy, its view-dependent error with respect to the reference mesh.

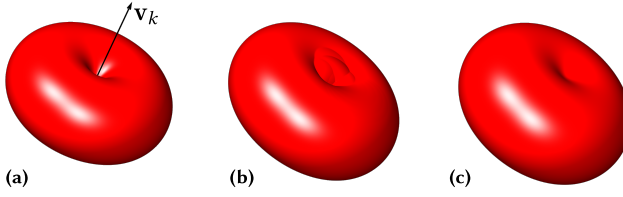


Fig. 2. Error metric visualization — we plot the error function  $E(\mathbf{d})$  for a single (a) and a set (b) of difference vectors, as well as its conservative approximation  $\tilde{E}(\mathbf{d})$ .

Since this error metric is the central ingredient of our framework, it is presented first in Section 4. The shared parametrization is used to measure the residual geometric and parametric distortions introduced by any simplification algorithm in a view-dependent fashion.

Starting from the given detailed level 0, the subsequent levels are obtained by an extended version of Lambert et al. [2016] decimation algorithm. We take advantage of our error metric to better drive the edge-collapses and to design a global optimization procedure that refits the vertex positions of the simplified mesh to the reference surface (Section 5). For each stored level, the view-dependent error of each patch is summarized by a conservative analytic approximation, enabling a compact storage and an efficient evaluation. At render time, they are used on the GPU to guide the patch level selection (see Figure 1b).

#### 4 VIEW-DEPENDENT ERROR METRIC

We describe in this section how we compute (Section 4.1) and summarize (Section 4.2), for each patch and at each level of the LOD, the error relative to the reference mesh in a view-dependent fashion so that adequate tessellation factors can be efficiently determined at render time (Section 4.3).

In the following,  $(\mathbf{p}_i, u_i, v_i)$  (resp.  $(\mathbf{q}_j, u_j, v_j)$ ) refers to the 3D position and texture coordinates of the  $i^{\text{th}}$  (resp.  $j^{\text{th}}$ ) vertex of the reference (resp. simplified) mesh.

##### 4.1 Mapping and Difference Vectors

We first focus on establishing a view dependent error metric for a single patch at a given level. For now, we only consider the impact of the view direction; the scaling factors due to the perspective projection and pixel resolution are taken into account at render time (Section 4.3). Our error metric is based on the bijection offered by the parametrization shared between the LOD and the reference mesh. Let  $\mathcal{M}_s : (u, v) \rightarrow (x, y, z)$  for the simplified patch and  $\mathcal{M}_r$  for the reference mesh be the functions which associate to every texture coordinates their corresponding 3D position. In particular,  $\mathbf{p}_i = \mathcal{M}_r(u_i, v_i)$  and  $\mathbf{q}_j = \mathcal{M}_s(u_j, v_j)$ . Taking the difference between the two surfaces in parametric space yields the vector-valued function

$$\mathcal{V}(u, v) = \mathcal{M}_s(u, v) - \mathcal{M}_r(u, v), \quad (1)$$

defining for every point a so called *difference vector*.

For a given unit view direction  $\mathbf{d}$ , the error introduced by a single vector  $\mathbf{v} = \mathcal{V}(u, v)$  is equal to the norm of its projection on screen, i.e.,  $\|\mathbf{v} \times \mathbf{d}\|$ . As depicted in Figure 2(a), if the view direction is aligned with the difference vector, the error vanishes. In contrast, the error becomes maximal when the view direction is orthogonal to the difference vector. For a patch and a direction  $\mathbf{d}$ , it is thus possible to compute a conservative error  $E(\mathbf{d})$ , that is equal to the maximum error introduced by all vectors for this direction:

$$E(\mathbf{d}) = \sup_{(u, v)} \|\mathcal{V}(u, v) \times \mathbf{d}\|. \quad (2)$$

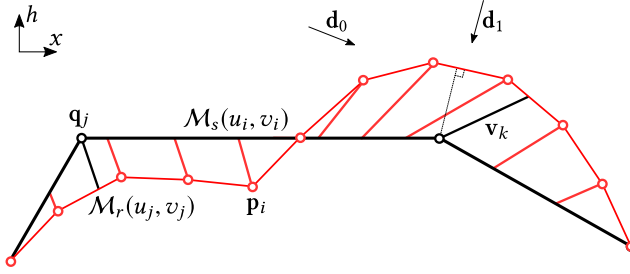


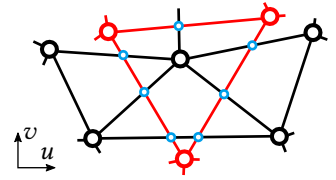
Fig. 3. Mapping —  $\mathcal{M}_s$  and  $\mathcal{M}_r$  allow to define the difference function  $\mathcal{V}$  between the reference mesh (in red) and the simplified mesh (in black) using their shared parametrization. As illustrated with the difference vector  $\mathbf{v}_k = \mathcal{V}(u_k, v_k)$ , this mapping generally differs significantly from the orthogonal projection of one mesh on the other. Also note how the error varies according to the view direction; for instance, it is much larger when seen from  $\mathbf{d}_0$  than from  $\mathbf{d}_1$ .

As  $\mathcal{M}_s$  and  $\mathcal{M}_r$  are continuous piecewise linear functions and  $\mathcal{V}$  is a linear combination of those, it is also continuous and piecewise linear. In addition, in equation (2), the cross product performs a linear combination of the components of  $\mathcal{V}(u, v)$ , and the norm transforms this continuous piecewise linear function into a continuous piecewise monotone function. Therefore, as we are only interested in maximum error values, we can evaluate  $E$  only for a discrete set of difference vectors  $\mathbf{v}_k \in V$  corresponding to the bounds of each linear piece:

$$E(\mathbf{d}) = \max_{\mathbf{v}_k \in V} \|\mathbf{v}_k \times \mathbf{d}\| . \quad (3)$$

In 2D, the linear pieces are segments bounded by the union of the vertices of the reference and simplified meshes. As illustrated in Figure 3, the difference vectors thus match the vertices  $\mathbf{q}_j$  of the simplified mesh with their image on the reference mesh  $\mathcal{M}_r(u_j, v_j)$ , and reciprocally the vertices  $\mathbf{p}_i$  of the reference mesh with their image on the simplified mesh  $\mathcal{M}_s(u_i, v_i)$ .

In 3D, the functions  $\mathcal{M}_s$  and  $\mathcal{M}_r$  are also linear on each triangular face of their respective domain. Thus the function  $\mathcal{V}$  is linear on each polygonal face formed by the intersection of the two meshes in parametric space, as depicted on the right inset figure. Consequently, to evaluate  $E$ , we need to consider the vectors originating from the vertices of the two meshes and from all edge intersections in parametric space.



Owing to the large number of difference vectors (in the order of  $10^4$ ), this equation is not suitable for fast evaluation. One can notice however that only a fraction of them are actually contributing to the *maximum* error, those belonging to the convex hull of the vector set (centered at the same origin); the other ones can be discarded. Although the number of vectors is much smaller after this pruning — from dozens to a few hundreds — it is still impractical for real-time GPU evaluation: a more compact representation is required.

## 4.2 Approximate error metric

To evaluate  $E$  efficiently, we need to find an approximation  $\tilde{E}(\mathbf{d}) \approx E(\mathbf{d})$  which is fast to evaluate and has a low memory footprint. Following the previous remark on the convex-hull reduction, a natural idea would be to further summarize the set of difference vectors by a simpler enclosing primitive

such as, for instance, an ellipsoid. However, evaluating the error represented by an ellipsoid in a direction  $\mathbf{d}$  involves a somewhat expensing eigenvalue extraction after a projection on a 2D plane. For the same reason, finding the enclosing ellipsoid that best approximate  $E$  for all view directions is a non trivial task. As demonstrated in Figure 4 (dashed curves), a simple heuristic such as taking the ellipsoid of minimal volume produces poor quality approximations.

We thus follows a more pragmatic approach that consists in directly approximating the error function  $E$  instead of approximating the difference vectors. To this end, let us make a few observations. First, we can observe in equation (3) that the error function  $E(\mathbf{d})$  is centrally symmetric:  $E(\mathbf{d}) = E(-\mathbf{d})$ . Second, in the case of a single difference vector  $\mathbf{v} = (v_x, v_y, v_z)$  and a direction  $\mathbf{d} = (x, y, z)$ , equation (2) becomes:

$$E(\mathbf{d}) = \|\mathbf{v} \times \mathbf{d}\| \\ = \sqrt{(v_y z - v_z y)^2 + (v_z x - v_x z)^2 + (v_x y - v_y x)^2},$$

which is the square-root of a quadratic trivariate polynomial limited to the second-order terms. A natural choice to approximate  $E(\mathbf{d})$  thus consists in seeking for functions  $\tilde{E}$  of the same form (see Figure 2), but with generalized coefficients:

$$\tilde{E}(\mathbf{d}) = \sqrt{a_1 x^2 + a_2 y^2 + a_3 z^2 + a_4 xy + a_5 xy + a_6 yz}. \quad (4)$$

We observe that since the vector  $\mathbf{d} = (x, y, z)$  is unitary, the function space of  $\tilde{E}^2$  is equivalent to the one defined by the constant and the five second-order basis functions of real spherical harmonics. Like  $E(\mathbf{d})$ , the function  $\tilde{E}(\mathbf{d})$  is by construction centrally symmetric. It only requires the storage of six scalar coefficients, and involves very few arithmetic operations to be evaluated.

To compute the best coefficients for  $\tilde{E}(\mathbf{d})$ , we first discretize the problem by taking a very large set of directions  $\mathbf{d}_i$  uniformly distributed on a sphere. In our experiments, we used  $10^4$  directions generated with a Spherical Fibonacci mapping [Keinert et al. 2015], which yields a nearly uniform distribution on the unit hemisphere (recall that  $E$  is centrally symmetric). Then, in order to linearize the problem, we minimize in the least-square sense the distance between the square of our metric  $\tilde{E}(\mathbf{d}_i)^2$  and the square error  $E(\mathbf{d}_i)^2$  for all directions. Moreover, we guarantee a conservative approximation by adding inequality constraints ensuring that for any direction,  $\tilde{E}$  is larger or equal to  $E$ . This boils down to a convex quadratic problem, which can be formally summarized as follows:

$$\begin{aligned} & \text{minimize} \quad \sum_i \left( \tilde{E}(\mathbf{d}_i)^2 - E(\mathbf{d}_i)^2 \right)^2 \\ & \text{subject to} \quad \tilde{E}(\mathbf{d}_i)^2 \geq E(\mathbf{d}_i)^2 \quad \forall \mathbf{d}_i. \end{aligned} \quad (5)$$

As visualized in Figure 2(c), this approximated error metric is indeed conservative.

*Evaluation.* To evaluate the accuracy of this approximation, we consider both the average  $A$  and maximum  $M$  error with respect to  $E$  over all directions on the unit sphere  $\Omega$ , normalized by the maximal value of  $E$ :

$$\begin{aligned} M &= \sup_{\mathbf{d}} (\tilde{E}(\mathbf{d}) - E(\mathbf{d})) / \sup_{\mathbf{d}} E(\mathbf{d}) \\ A &= \frac{1}{4\pi} \int_{\Omega} (\tilde{E}(\mathbf{d}) - E(\mathbf{d})) / \sup_{\mathbf{d}} E(\mathbf{d}). \end{aligned}$$

For comparison purpose, we also compute these indicators with an isotropic version of our error metric, defined as:

$$E_{iso} = \sup_{\mathbf{d}} E(\mathbf{d}) = \max_k \|\mathbf{v}_k\|.$$



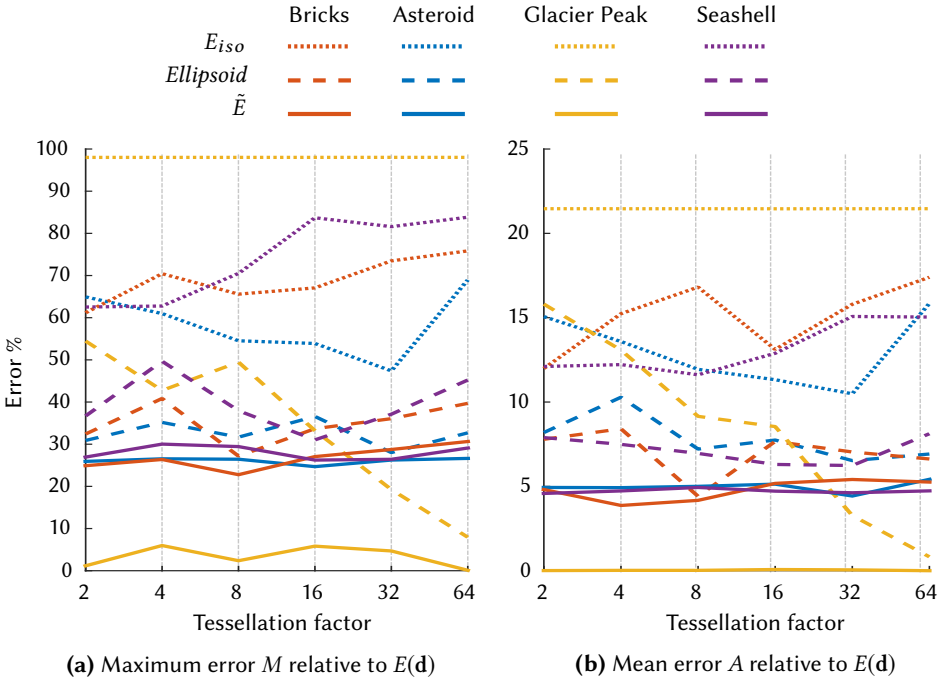


Fig. 4. Error metric comparison — for each 3D model of Figure 11, at each tessellation factor, the maximum (a) and mean (b) error of three approximate error metrics have been computed and averaged over all patches generated using the simplification algorithm described in Section 5.

The normalization by the maximal value of  $E$  compensates for large variations of the magnitude of  $E$  across levels, patches and meshes. These measures are summarized in Figure 4 for four different meshes depicted in Figure 11. In these plots, for each level we take the maximum of  $M$  and average of  $A$  over all patches of the given mesh.

Our conservative metric  $\tilde{E}$  clearly outperforms the isotropic approximation. In the worst case, our metric overestimates the actual LOD deviation by about 30%, while the isotropic error overestimation ranges from 50 to 90% (Figure 4a). The approximation provided by the ellipsoids of minimal volume is somewhat in-between, but is less stable across meshes. Our metric can suffer some overestimation because the quadratic polynomial fails to represent well multiple minima, which can happen when the error is small for multiple view directions. Yet the average error is very small, about 5%, compared to 15-20% with the isotropic metric and 5-10% with the ellipsoids (Figure 4b). This is especially noticeable for height-field models such as the “Glacier Peak” terrain, for which an almost unique direction is favored (the direction of elevation) at every level. This demonstrates that our approximation is very accurate in most cases, and that a more sophisticated approximation could only yield very small gains.

### 4.3 GPU Evaluation

Once the coefficients of the error metric  $\tilde{E}$  have been computed for all patches at all levels, they are stored in a 1D texture buffer. For power-of-two levels, this implies a memory cost of 6 levels  $\times$  6 coefficients  $\times$  4 bytes = 144 bytes per patch. The LOD meshes are stored similarly to Lambert et al. [2016], which requires 36 bytes per vertex of the reference mesh and intermediate power-of-two levels. The memory overhead of storing the error metric coefficients is thus marginal.

Our rendering pipeline then follows the standard hardware tessellation steps. For each patch, the tessellation factor is computed in the evaluation control shader according to the current camera position and orientation (see Figure 1b). More precisely, the direction of evaluation  $\mathbf{d}$  is computed as the normalized difference between the camera and patch center position. Since our metric is defined in object space, we further apply the camera rotation and object transformation to this direction. Then, starting from the lowest tessellation level, we iteratively fetch the coefficients of the quadratic polynomial using the *primitive ID* of the patch, and we evaluate the metric using equation (4) until the error is inferior to a user-defined pixel bound. To take into account perspective distortions, and hence the viewing distance, we project the error centered on the patch orthogonally to the view direction, making the assumption that the scaling factor and view-direction are constant per patch. This assumption does not hold for very close viewpoints but, in such cases, the distance to the camera dominates the directionality of the metric.

To guarantee that adjacent patches are seamlessly connected, patch-border tessellation factors are set to the maximum error of the two patches. The interior tessellation factor is then defined as the maximum factor of the three borders.

## 5 LOD GENERATION

Our LOD generation pipeline is based on the recent simplification method of Lambert et al. [2016] which works as follows (see Algorithm 1 and Figure 5). It takes as input a coarse mesh made of triangular patches tessellated at the maximum factor  $f_{\max}$  (64 on current GPUs). This mesh is then progressively simplified by iteratively collapsing for each patch corner the strip of edges that minimize a given edge selection cost. More precisely, for a given patch corner, step 11 starts by updating the contraction cost  $C_{edge}$  of all the edges that may belong to the associated strip. Similarly to previous work, this is achieved by virtually contracting each considered edge using either the QEM [Garland and Heckbert 1997] or LTM [Lindstrom and Turk 1998]. The strip whose sum of edge costs is minimal is then found in linear time using a variant of Dijkstra algorithm over a predefined binary directed acyclic graph (DAG). Finally, in step 12, the edges of the best strip are contracted one after the other. After each iteration, the tessellation factor of the mesh is decremented by two and its topology is guaranteed to match the hardware tessellation pattern. In order to balance storage cost and LOD granularity, only levels with a power-of-two tessellation factor are stored, as recommended by Lambert et al. [2016].

In this section, we present how our novel view-dependent error metric can guide this simplification, by extending the LOD generation in three ways. First, we derive a global optimization step which aims at refitting the simplified mesh to the reference model once a given LOD is reached (step 4, Section 5.1). Second, we propose a new placement strategy that computes jointly the position and texture coordinates of the merged vertices (step 12, Section 5.2). Finally, we describe a new edge contraction cost  $C_{edge}$  based on the current view-dependent error (step 11, Section 5.3).

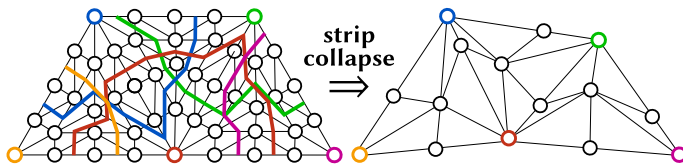


Fig. 5. Strip-based simplification — at each iteration, for each patch corner, one strip of edges is contracted decrementing by two the tessellation factor.

**Algorithm 1** Strip-based mesh simplification — our extensions to [Lambert et al. 2016] are highlighted in red.

---

```

1: Input: tessellated coarse mesh at factor  $f \leftarrow f_{\max}$  + reference mesh
2: loop
3:   if  $f$  is power-of-two then ▷ desired tessellation level
4:     Refit current mesh to the reference mesh. ▷ 5.1
5:     Compute  $\tilde{E}$  for each patch. ▷ 4.2
6:     Store attributes for the current level.
7:   end if
8:   if  $f = f_{\min}$  then break.
9:   for each patch corner do
10:    Update  $\tilde{E}$  for each adjacent patch.
11:    Find the strip minimizing the edge selection cost  $C_{edge}$ . ▷ 5.3
12:    Contract the edges of the strip and compute texture coordinates. ▷ 5.2
13:  end for
14:   $f \leftarrow f - 2$ 
15: end loop

```

---

## 5.1 Refitting

Our goal here is to globally optimize the vertex positions such as to reduce the error  $E(\mathbf{d})$  for all patches and any view direction  $\mathbf{d}$  (step 4 of Algorithm 1). Since it would be intractable to directly minimize our metric  $E$ , we devise a related but simpler optimization problem. Let us first observe that the error  $E$  is highly related to the magnitudes of the difference vectors. Therefore, reducing them should also reduce  $E$ . To this end, we assume that the texture coordinates  $(u_j, v_j)$  of the vertices of the current simplified mesh are fixed, and we seek to find their respective 3D positions  $\mathbf{q}_j$  that minimize the length of the difference vectors  $\mathbf{v}_k$  of  $V$ , i.e., that minimize the average distance between the simplified and reference meshes in the least-square sense. This can be formulated as the following weighted least-square optimization problem:

$$\min \sum_{\mathbf{v}_k \in V} w_k \|\mathbf{v}_k\|^2, \quad (6)$$

where, for now,  $w_k = 1$ , and  $V$  is the set of all difference vectors linking the reference and simplified meshes, as defined in Section 4.2. For a given vertex  $(\mathbf{p}_i, u_i, v_i)$  of the reference mesh, the respective difference vector  $\mathcal{V}(u_i, v_i)$  going to the point  $\mathcal{M}_s(u_i, v_i)$  of the simplified mesh can be expressed using barycentric coordinates as:

$$\begin{aligned} \mathcal{V}(u_i, v_i) &= \mathcal{M}_s(u_i, v_i) - \mathbf{p}_i \\ &= \sum_{j \in T_i} \lambda_j \mathbf{q}_j - \mathbf{p}_i. \end{aligned}$$

The barycentric coordinates  $\lambda$  are computed in parametric space by finding the triangle  $T_i$  of the simplified mesh that contains the point  $(u_i, v_i)$ , such that  $(u_i, v_i) = \sum_{j \in T_i} \lambda_j (u_j, v_j)$ . Reciprocally, for a vertex  $(\mathbf{q}_j, u_j, v_j)$  of the simplified mesh, we get:

$$\mathcal{V}(u_j, v_j) = \mathbf{q}_j - \sum_{i \in T_j} \lambda_i \mathbf{p}_i.$$

Using the same logic for the difference vectors corresponding to the edge intersections, we end up with a very sparse linear system where the components  $x, y, z$  of the unknown  $\mathbf{q}_j$  are independent.

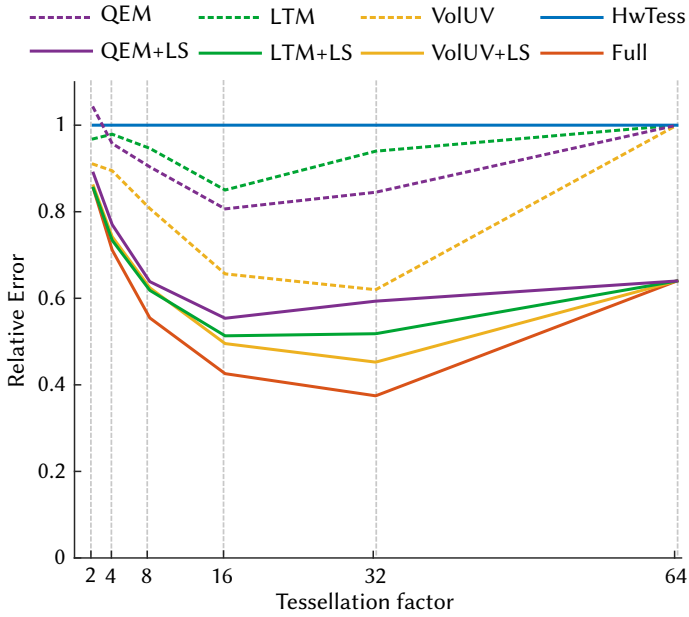


Fig. 6. Simplification algorithms comparison — the error is expressed relatively to the mesh produced by the regular hardware tessellation (HwTess) and averaged over all patches of the four 3D models shown in Figure 11. We compare the QEM- and LTM-based vertex placement strategies with our new Volumetric UV optimization with and without least-square refitting, and eventually its *full* combination with our novel edge selection cost.

In our implementation, we solve this system through the normal equation and the direct sparse Cholesky solver of the Eigen library [Guennebaud et al. 2016].

The error  $E$  can be further reduced by observing that  $E$  is mostly influenced by the longest difference vectors, whereas reducing the length of the shorter vectors, that are unlikely to take part in the convex hull of  $V$ , would leave the global error unchanged. In order to take into account this behavior while keeping a simple and fast optimization procedure, we extend the above least-square optimization by iteratively re-weighting the difference vectors according to their ratio to the largest vector:

$$w_k = \gamma + \frac{\|\mathbf{v}_k\|}{\sup_{\mathbf{v}_i \in V} \|\mathbf{v}_i\|}, \quad (7)$$

where  $\gamma$  is a small constant to avoid close to zero weights for small vectors ( $\gamma = 0.02$  in all our experiments). In practice, after a first refitting iteration with unit weights, we apply a few passes of weighted refitting until the mean error over all directions stops decreasing. Note that, at each iteration, we only need to recompute the weights; the above barycentric coordinates only need to be computed once.

This iteratively re-weighted refitting process is applied each time a tessellation level is stored (step 4 in Algorithm 1). To avoid cracks between adjacent patches tessellated at different levels, the position and texture coordinates of the patch corners cannot change during the simplification. Consequently, they are only optimized during the initial refitting step of the level 0 and considered fixed during the subsequent ones.

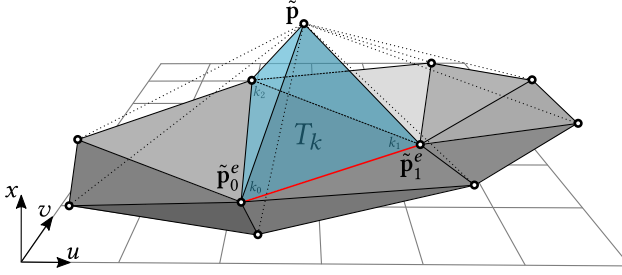


Fig. 7. Construction of our vertex placement metric – the two vertices  $\tilde{p}_0^e$  and  $\tilde{p}_1^e$  are collapsed into the merged vertex  $\tilde{p}$ . For each of the three position components, here  $x$ , each adjacent face  $T_k = (k_0, k_1, k_2)$  is connected to  $\tilde{p}$  to form tetrahedrons in the  $(u, v, x)$  space (one of them is highlighted in blue). Their unsigned volumes are minimized in the least-square sense to find both the 3D position and  $uv$ -coordinates of  $\tilde{p}$ .

*Evaluation.* The impact of this refitting step is evaluated in Figure 6 using the integral of our error metric  $E$  as an objective comparison criterion. For this experiment, we compare both the QEM- and LTM-based vertex placement strategies relatively to the LOD produced by the classic hardware tessellation pipeline (i.e., by sampling mipmapped displacement maps). Since we require the interpolation of texture coordinates, we use the QEM variant proposed by Lambert et al. [2016], that constrains the vertex placement on the collapsed-edge with linear texture coordinates interpolation, and used a simple Shepard interpolation [Shepard 1968] of the neighboring vertex texture coordinates for the LTM. To ensure a fair comparison, all the methods start with the same initial tessellated mesh (level 0 in Figure 1) obtained by sampling an intermediate displacement map. This explains why, without refitting, they all exhibit the exact same error for the highest tessellation factor. As expected, for both placement strategies, this refitting step has a huge impact on the quality of the produced LOD with an advantage for the LTM (solid green curve in Figure 6).

## 5.2 Vertex placement strategy

In this section we revisit the local vertex placement strategy applied during the contraction of an edge (step 12 in Algorithm 1) taking advantage of our global refitting procedure and the insights we developed. Since the refitting step takes care of recomputing the vertex positions solely from the texture coordinates of the simplified mesh, the role of the vertex placement is slightly shifted. Its main goal is to compute as accurate as possible texture coordinates, while keeping good enough vertex positions for dependent edge contractions to be properly carried out. In particular, the volume preservation property of all previous simplification strategies is not required anymore.

To this end, we propose a new placement strategy which optimizes jointly the position and texture coordinates of the merged vertices. Our approach borrows the unsigned volume optimization constraint of Lindstrom and Turk [1998] but expresses it in a mixed texture and position space. Our construction is depicted in Figure 7. For each face  $T_k = (k_0, k_1, k_2)$  adjacent to the collapsed vertices  $p_0^e$  and  $p_1^e$ , we derive three volume minimization objectives, one for each position component, by constructing tetrahedrons from the position component and the two  $uv$ -coordinates of the vertices.

Let  $\tilde{p} = (x, y, z, u, v)$  be the unknown position and texture coordinates of the merged vertex. For the  $x$  component, the unsigned volume objective can be written as:

$$f_x(\tilde{p}) = \sum_{T_k} \left( [u \ v \ x] \mathbf{n}_{T_k}^{uvx} - \begin{bmatrix} u_{k_0} & u_{k_1} & u_{k_2} \\ v_{k_0} & v_{k_1} & v_{k_2} \\ x_{k_0} & x_{k_1} & x_{k_2} \end{bmatrix} \right)^2 \quad (8)$$

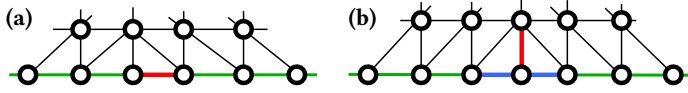


Fig. 8. Edge-collapses involving a patch boundary (green) — the position of the merged vertex is either constrained **(a)** on the collapsed-edge (red), or **(b)** on the two adjacent boundary edges (blue).

where  $\mathbf{n}_{T_k}^{uvx}$  is the unnormalized normal of triangle  $T_k$  obtained as the cross product of two edges in the  $(u, v, x)$  space. Deriving similar objectives for the  $y$  and  $z$  components and summing them, the full objective function is:

$$f(\tilde{\mathbf{p}}) = f_x(\tilde{\mathbf{p}}) + f_y(\tilde{\mathbf{p}}) + f_z(\tilde{\mathbf{p}}) + \epsilon \left\| \tilde{\mathbf{p}} - \frac{\tilde{\mathbf{p}}_0^e + \tilde{\mathbf{p}}_1^e}{2} \right\|^2, \quad (9)$$

where the last regularization term attracts the new vertex towards the center of the collapsed edge when all the adjacent faces become coplanar in the five dimensional space. To make this term scale-independent, we normalize the positions and texture coordinates by the length of the diagonal of the bounding box enclosing the adjacent faces in 3D space and parametric space respectively. This non-uniform scaling further allows us to set the  $\epsilon$  factor once and for all, irrespectively of the input mesh. To this end, we analyzed the numerical conditioning of this energy without the regularization term for typical mesh configurations, and empirically we found that taking  $\epsilon = 10^{-4}$  is a good trade-off between the placement flexibility and numerical stability.

Associated to this new placement strategy, the natural cost for collapsing the corresponding edge is the sum of the three unsigned volume objectives  $f_x$ ,  $f_y$  and  $f_z$ , i.e., the residual of equation (9) without the aforementioned regularization and rescaling. To the best of our knowledge, this is the first edge-collapse cost function taking into account both the geometric and parametric distortions in a scale invariant manner.

*Discussions.* This objective function shares strong similarities with the least-square optimization derived in the previous section: for a single position component, minimizing the unsigned volumes of the attached tetrahedrons corresponds to minimizing the continuous integral of the difference function  $\mathcal{V}$  computed between the two meshes before and after the contraction. Even though the difference vectors are not taken relative to the reference mesh, the relation established in the previous section with the minimization of our error metric  $E$  still holds here. It is also worth mentioning that the position components are not independent anymore since they are linked by the unknown texture coordinates.

*Patch boundaries.* The GPU tessellation requires that vertices of the patch boundaries remain on the boundary edge, otherwise artifacts may appear when a boundary edge receives a different tessellation level than the patch interior. However an edge-collapse can affect a patch boundary in two situations (red edges in Figure 8): (1) when contracting an edge belonging to the boundary, and (2) when collapsing an edge that joins a vertex on the boundary and a vertex on the patch interior. In both cases, the texture coordinates of the merged vertex needs to be a linear interpolation of two vertices on the boundary. There is only one possibility in the former configuration, but two in the latter as depicted by the blue edges in Figure 8b. In this case, we evaluate the solution for both edges, and keep the one with minimal residual. Moreover, to handle degenerated cases in this particular configuration, we use a modified regularization term in equation (9) that attracts the merged vertex toward the edge extremity which is on the patch boundary.

In practice, to constrain the texture coordinates on an edge, the unknowns  $u$  and  $v$  are replaced by a new unknown  $\alpha$  such that:

$$[u, v] = \alpha[u_0^e, v_0^e] + (1 - \alpha)[u_1^e, v_1^e] \quad \text{with } 0 \leq \alpha \leq 1.$$

With such a formulation, the objective function of equation 9 leads to a quadratic problem that can be solved similarly for patch boundaries, open mesh boundaries and texture coordinate discontinuities.

*Evaluation.* Our new vertex placement strategy is referenced as *VolUV* in Figure 6 (yellow curves), for *Volumetric UV* optimization. It can be seen that this new strategy alone, i.e., without the previously described global refitting step, already outperforms both the QEM and LTM strategies. Once combined with the least-square (LS) refitting, the higher accuracy of the texture coordinates computed with our method leads to substantial improvements over LTM+LS.

### 5.3 Edge-collapse Selection Strategy

All edge-collapse simplification algorithms use an edge selection strategy to prioritize contractions. The strategy that we just defined in Section 5.2 is already a great improvement over prior work, but it remains very local and it is only indirectly related to our patch-based metric  $E$ . To get closer to the minimization of  $E$ , we propose to modify this strategy to favor edge-collapses that do not increase  $E$  from any view direction, and thus results in a better distribution of the error over the entire patch (step 11 of Algorithm 1).

Each contraction modifies the difference vectors in the one-ring neighborhood of the collapsed-edge. Using the mappings  $\mathcal{M}_r$  and  $\mathcal{M}_s$  between the reference and simplified mesh, we can efficiently extract these difference vectors (see Section 6 for details). The natural solution to evaluate whether the error metric would be increased by this contraction would be to compute the view-dependent metric before and after each tentative edge-collapse, but repeatedly solving the convex quadratic problem of equation (5) for each edge would be prohibitively expensive. We thus propose the following alternative.

First, before each strip-collapse, we ensure that the coefficients of our analytic error metric  $\tilde{E}_{curr}$  is up-to-date for each considered patch using equation (5) (step 10 of Algorithm 1). Since the number of strip-collapses is orders of magnitude smaller than the number of edge-collapses, this is a reasonable compromise. Then, for each tentative edge-collapse, following equation (3), we define the error

$$E_{col}(\mathbf{d}) = \max_{\mathbf{v}_k \in V_{edge}} \|\mathbf{v}_k \times \mathbf{d}\|$$

introduced by the difference vectors  $V_{edge}$  affected by this contraction. To speed up the computation, the set  $V_{edge}$  is again reduced to its convex hull, as in Section 4.2. Finally, we compare  $E_{col}$  with  $\tilde{E}_{curr}$  for a reasonably small set of directions  $D$ , and average the positive differences, yielding the following cost:

$$C_{edge} = \frac{1}{|D|} \sum_{\mathbf{d} \in D} \max(0, E_{col}(\mathbf{d}) - \tilde{E}_{curr}(\mathbf{d})). \quad (10)$$

Negative differences must be ignored because they only imply that the corresponding vectors are not contributing to the error in this direction, not that the error  $E$  would decrease. In our experiments we used 200 directions.

Then, in step 11 of Algorithm 1, we search for the strip in the binary DAG whose sum of edge costs is the lowest. At each node of the graph, if the two possible paths have the same cost, which is often the case when contractions do not increase the error in any direction (i.e.,  $C_{edge} = 0$ ), we fall back on the standard edge selection cost of the respective simplification algorithm. In practice it implies that both costs need to be computed and propagated.

*Evaluation.* As illustrated on the plots of Figure 6 (red curve), this selection strategy, applied in conjunction with the two previous extensions, hence called *Full*, allows to further reduce the view-dependent error by up to 60% compared to the regular hardware tessellation.

## 6 IMPLEMENTATION DETAILS

### 6.1 Level 0 initialization

To ensure that the first level in the LOD hierarchy matches the fixed hardware tessellation pattern, we take as input the detailed reference mesh and a coarse decomposition into triangular patches sharing the same parametrization. The decomposition is either obtained from the modeling process itself, or a posteriori using any decimation algorithm that preserves the texture coordinates.

The initial level 0 is then generated by instantiating each patch of the coarse mesh at the highest tessellation factor ( $f_{\max} = 64$  for current GPU) with texture coordinates linearly interpolated from the patch corners. The first refitting process (step 4 of Algorithm 1) will take care of computing the respective 3D positions. Better starting points could be obtained by non-uniformly spreading the vertices in parametric space [Yuan et al. 2016], but we did not investigate such an approach yet.

### 6.2 Computation of difference vectors

*Vertex to mesh mappings.* In Section 4.1, for a given point  $\mathbf{p}_i$  on the reference mesh (resp.  $\mathbf{q}_j$  on the simplified mesh), we need to efficiently find the point  $\mathcal{M}_s(u_i, v_i)$  on the simplified mesh with the same texture coordinates (resp.  $\mathcal{M}_r(u_j, v_j)$  on the reference mesh). To accelerate this search, we use two 2D AABB-trees built in parametric space over the reference and simplified meshes. As the former is constant, its associated AABB-tree only needs to be computed once and is used thorough the simplification process. The latter, however, would need to be updated after each edge-collapse.

In our implementation, this AABB-tree is used only once to initialize, for each face of the simplified mesh, the list of the reference mesh vertices that map to this face. This list is then quickly updated after each edge-collapse by re-assigning the difference vectors in the one-ring neighborhood of the collapsed-edge to the new adjacent faces of the merged vertex. In the same vein, the point  $\mathcal{M}_r(u_j, v_j)$  on the reference mesh corresponding to the vertex  $\mathbf{q}_j$  of the simplified mesh can be cached and computed for each newly merged vertex using the AABB-tree. This custom data structure also permits to very efficiently evaluate the local error metric  $E_{col}$  in Section 5.3 as the difference vectors affected by a given edge-collapse are readily available from the adjacent faces lists.

*Edge-edge intersections in parametric space.* As explained in Section 4.2, to compute a conservative error metric, we also need to consider the difference vectors occurring at edge-edge intersections in the 2D parametric space. Those are computed using a 2D AABB-tree built over the edges of the reference mesh in parametric space. Since this a somewhat costly operation, these intersections are computed for power-of-two levels only, that is, before the global refitting and computation of the error metric  $\bar{E}$  (steps 4 and 5 of Algorithm 1). We emphasize that since these two steps exploit the same set of difference vectors, they can thus be established only once for both. In contrast, in Section 5.3, difference vectors originating from such edge-edge intersections are ignored; only the intersections originating from the vertices of the reference and simplified meshes are considered for both the update of  $\bar{E}$  in step 10 of Algorithm 1 and the evaluation of  $E_{col}$  in equation (10).

## 7 RESULTS

We evaluated the performance of our method on a Intel i7-4790K @ 4GHz CPU with a Nvidia Geforce 980 GTX. In our experiments, the target tolerance error for the LOD selection has been set to 1 pixel for a window resolution of  $1920 \times 1140$  pixels.



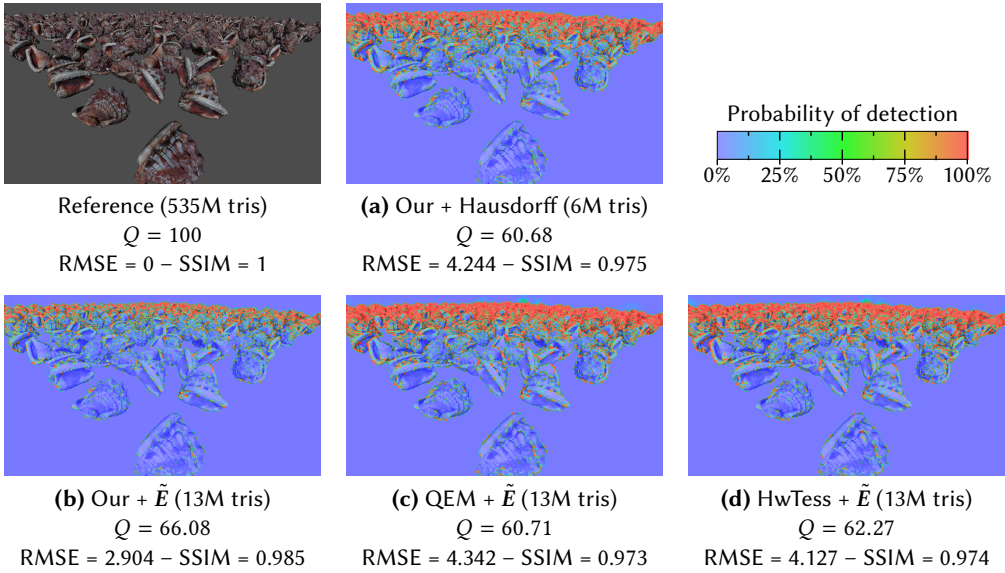


Fig. 9. Comparison with the perception-correlated image quality metric HDR-VDP-2 [Mantiuk et al. 2011] – our LOD generation method combined with our anisotropic metric (b) produces the lowest amount of distortions (larger quality prediction  $Q$ ) compared to (a) a pure geometric selection metric, (c) a geometric-based LOD generation, and (d) standard displacement mapping. The same conclusion can be drawn when comparing the Root Mean Square Error (RMSE) and Structural Similarity Index (SSIM) [Wang et al. 2004].

## 7.1 Subjective evaluation

Figure 9 provides a subjective evaluation of our selection metric and generation algorithm using a perception-correlated image metric [Mantiuk et al. 2011] as well as the Root Mean Square Error (RMSE) and Structural Similarity Index (SSIM) [Wang et al. 2004]. In this figure we first compare our anisotropic selection metric  $\tilde{E}$  (b) to a selection based on the Hausdorff distance (a). In both cases, we set the tolerance to 1 pixel. Unsurprisingly, since the Hausdorff distance ignores parametric distortions, coarser levels have been selected yielding to much stronger perceptually visible distortions in the rendered image. The other two images aim to compare our new LOD generation procedure to the QEM variant of Lambert et al. [2016] (c), and to regular hardware tessellation with mip-mapped displacement maps (d). For this experiment, the tolerance threshold of our selection metric has been adjusted to produce the same triangle budget than in (b). In the supplemental materials, we also compare our anisotropic metric  $\tilde{E}$  with the isotropic version  $E_{iso}$ ; the former systematically outperforms the latter.

Even though we designed our metric and LOD generation procedure in a conservative and objective manner, the results of these three metrics are consistent with the results of Figure 6: for the same triangle count, perceptual image distortions are much lower for our method.

## 7.2 View-Dependent Metric Performance

Figure 10 compares the performance of our selection metric  $\tilde{E}$  with the isotropic version  $E_{iso}$  on the four test scenes of Figure 11 for three LOD generation approaches: the regular hardware tessellation with mip-mapped displacement maps, the QEM-based simplification method of Lambert et al. [2016], and our novel algorithm. Since the goal of our LOD selection strategy is to choose the appropriate level such that there is no visible difference with the reference mesh, an error

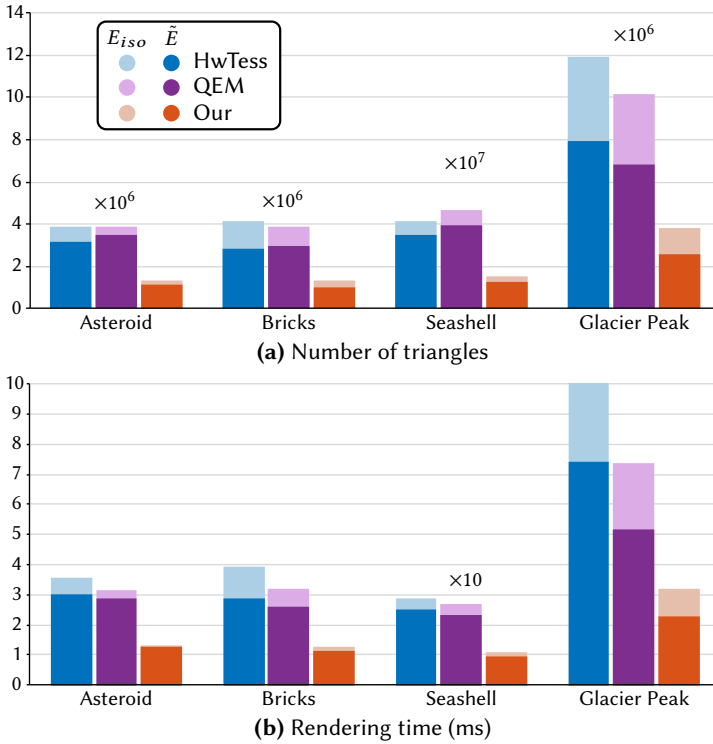


Fig. 10. Rendering performance comparison — for a given viewpoint, we measure the number of triangles (a) after level selection using our view-dependent metric  $\tilde{E}$  and the isotropic version  $E_{iso}$ , as well as the associated rendering time (b). We compare these indicators for three LOD generation methods on the four test meshes of Figure 11 instantiated 400 times on a 2D grid with random orientations.



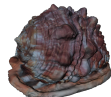

	Asteroid	Bricks	Seashell	Glacier Peak
reference mesh				
# triangles	491k	786k	1474k	786k
# patches	20	32	60	32

Fig. 11. Test scenes used for the evaluation.

threshold of 1 pixel was used for all methods. On all test scenes, instantiated 400 times on a uniform  $20 \times 20$  2D grid with random orientations, for a given LOD generation method, our view-dependent selection metric reduces the number of polygons generated by the tessellator by an average of 10% to 35% at equal visual quality. When comparing the LOD generation techniques with each others, our method reduces drastically the number of required triangles compared to previous work, accelerating the rendering time by a factor ranging from 2 to 3 on average. On the other hand, when using our parametrization-aware selection metric, Lambert et al.’s method (QEM) and regular HW tessellation exhibit similar performance. This because QEM optimizes for geometry only, whereas regular tessellation naturally preserves parametrization.

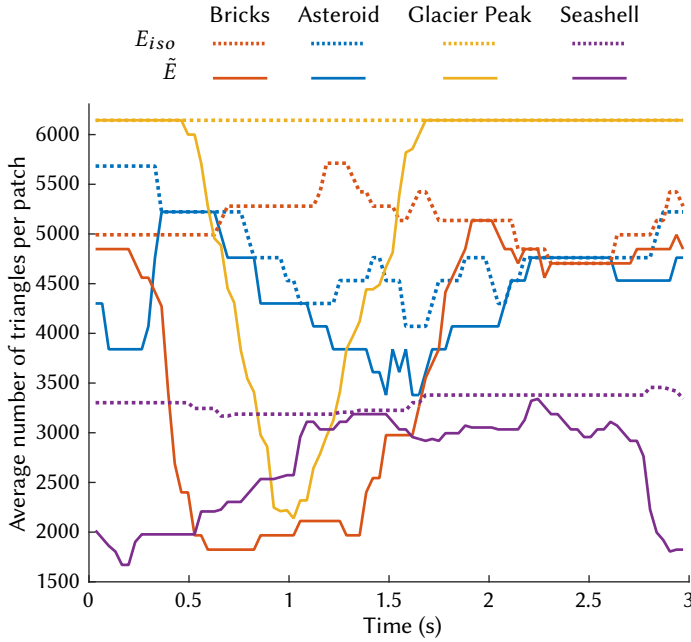


Fig. 12. Average number of triangles per patch along predefined view paths for our four reference models and for both the isotropic and our view-dependent error selection metric.

The view-dependent nature of our selection metric is better demonstrated in Figure 12, which reports the number of rendered triangles when rotating around a single instance of the previous 3D objects (see the accompanying video). Highest rate variations are observed for the “Bricks” and “Glacier Peak” models that both exhibit a privileged displacement direction: the error introduced by the simplification is thus strongly anisotropic, which is especially well captured by our metric. In both cases, the number of rendered triangles is maximal for grazing view directions, and minimal for nearly orthogonal view directions (between 0.5 and 1.5s). Note that unlike the “Bricks” model, the “Glacier Peak” model is an extreme case where the  $x$  and  $y$  coordinates of the input reference mesh matches with the  $uv$ -coordinates. Even though the other two 3D models are closed meshes, relatively high triangle count variations (roughly  $\times 2$ ) are still observed as the amount of geometric details is not evenly distributed among the different patches. Compared to the isotropic selection metric, the gain in triangle count range from 0% for some rare view directions up to 66%. Comparing the plots of the two metrics, we can estimate whether the variation in triangle count comes from the change of distance or orientation during the rotation. The plot of the isotropic metric for the “Seashell” model is flatter than for the others because we selected a more distant point of view: the relative distance of the patch to the camera is thus more constant during the rotation.

### 7.3 Pre-computation times

The pre-computation times of our LOD generation procedure is reported in Table 1 for the same four meshes with the breakdown for the main components of our algorithm. The column  $VolUV$  represents the cost of the overall generation process if the edge-edge intersections, refitting, and our edge-collapse cost (equation (10)) are omitted. It includes the initialization and propagation of the vertex-to-mesh mappings as well as the resolution of the QP problems to compute  $\tilde{E}$ . This part of the algorithm is rather fast, and the LS refitting step alone adds only a small overhead

Mesh	Total	VolUV + $\bar{E}$	Edge-edge intersec.	Refitting	$C_{edge}$
Asteroid	176	10	2	6	158
Bricks	327	17	6	10	294
Seashell	737	35	18	21	663
Glacier Peak	307	17	6	9	275

Table 1. Pre-computation time (in seconds) of our full LOD generation method and its main components.

to the generation process. In contrast, the computation of the edge-edge intersections and our edge selection cost  $C_{edge}$  greatly dominate the overall cost. However, our edge-edge intersection implementation is not optimized at all, and we believe that its cost could be reduced by one or two orders of magnitude using a more advanced algorithm and implementation. A similar gain could be obtained for  $C_{edge}$  by computing this cost in parallel for each edge, and exploiting SIMD instruction sets to evaluate  $E_{col}$  over hundreds of directions.

## 8 PERSPECTIVES AND CONCLUSIONS

In the first part of this paper, we introduced a new view-dependent metric capturing both geometric and parametric distortions between corresponding regions of two meshes. This metric can be used to objectively compare simplification algorithms, but thanks to its view-dependent nature and the lightweight approximation we proposed, it is especially useful to drive the render-time LOD selection in the context of patch-based GPU tessellation, with significant gains compared to an isotropic metric.

In the second part of this paper, we built upon the insights gained by this metric to revisit classical edge-collapse simplification algorithms and presented: (1) a new scale-invariant local vertex-placement strategy and cost metric taking into account vertex positions and textures coordinates, (2) a global iterative refitting procedure, and (3) an edge-collapse selection cost directly driven by our view-dependent metric. Their integration within a LOD generation pipeline compatible with hardware tessellation led to huge improvements both in terms of triangle count and rendering time.

Although our edge-collapse selection cost is dedicated to patch-based simplification algorithms, we emphasize that both our new vertex-placement strategy and refitting procedures could be integrated within classical edge-collapse simplification algorithms. The integration of the former is straightforward, and refitting passes could be triggered on a regular basis during the simplification. If this combination is as successful as in our HW tessellation context, this would lead to a highly competitive tool for the decimation of parametrized meshes.

Our LOD selection metric has been designed with rigidly deformed objects in mind. For reasonably small deformations, transforming our view-depend metric by a local rigid approximation on a patch-basis is likely to behave well. Handling large deformation, however, remains an open problem for future work.

Finally, our LOD generation pipeline and selection metric reached a rather high level of sophistication, and it is unclear how many additional triangles could still be saved by further intricacy. We believe that the highest gains could be obtained by (1) investigating better algorithms for the generation of the initial finer level, which is currently uniformly sampled in the parametric space, and (2) by taking into account the visibility (backfacing triangles and self-occlusions), the lighting conditions and their interaction with the materials to design a more perceptually-driven approach, even though this would likely significantly increase the dimension of the error metric.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their comments and suggestions. This work has been supported by the ANR *RichShape* project (ANR-14-CE24-0004).

## REFERENCES

- Iain Cantlay. 2001. *DirectX 11 Terrain Tessellation*. Technical Report. [https://developer.nvidia.com/sites/default/files/akamai/gamedev/files/sdk/11/terraitessellation\\_whitepaper.pdf](https://developer.nvidia.com/sites/default/files/akamai/gamedev/files/sdk/11/terraitessellation_whitepaper.pdf)
- Andrea Ciampalini, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. 1997. Multiresolution decimation based on global error. *The Visual Computer* 13, 5 (1997), 228–246.
- Paolo Cignoni, Claudio Montani, and Roberto Scopigno. 1998a. A comparison Paolomesh simplification algorithms. *Computers & Graphics* 22, 1 (1998), 37–54.
- Paolo Cignoni, C. Rocchini, and Roberto Scopigno. 1998b. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum* 17, 2 (1998), 167–174.
- Jonathan Cohen, Marc Olano, and Dinesh Manocha. 1998. Appearance-preserving Simplification. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM, 115–122.
- Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. 1996. Simplification Envelopes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. ACM, 119–128.
- Massimiliano Corsini, Mohamed-Chaker Larabi, Guillaume Lavoué, Oldrich Petřík, Libor Váša, and Kai Wang. 2013. Perceptual metrics for static and dynamic triangle meshes. *Computer Graphics Forum* 32, 1 (2013), 101–125.
- Evgenij Derzaf and Michael Guthe. 2012. Dependency-Free Parallel Progressive Meshes. *Computer Graphics Forum* 31, 8 (2012), 2288–2302.
- Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH '97)*. 209–216.
- Michael Garland and Paul S. Heckbert. 1998. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings of the 9th IEEE Visualization Conference (VIS '98)*.
- Gaël Guennebaud, Benoît Jacob, et al. 2016. Eigen v3. <http://eigen.tuxfamily.org>. (2016).
- Hugues Hoppe. 1997. View-Dependent Refinement of Progressive Meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH '97)*. 189–198.
- Hugues Hoppe. 1999. New Quadric Metric for Simplifying Meshes with Appearance Attributes. In *Proceedings of the 10th IEEE Visualization Conference (VIS '99)*. IEEE.
- Hanyoung Jang and Junghyun Han. 2013. GPU-optimized indirect scalar displacement mapping. *CAD Computer Aided Design* 45, 2 (2013), 517–522.
- Benjamin Keiwert, Matthias Innmann, Michael Sängler, and Marc Stamminger. 2015. Spherical Fibonacci Mapping. *ACM Transactions on Graphics* 34, 6 (2015), 193:1–193:7.
- Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. 1996. Mesh Reduction with Error Control. In *Proceedings of the 7th IEEE Visualization Conference (VIS '96)*. IEEE, 311–318.
- Thibaud Lambert, Pierre Bénard, and Gaël Guennebaud. 2016. Multi-Resolution Meshes for Feature-Aware Hardware Tessellation. *Computer Graphics Forum* 35, 2 (2016), 253–262.
- Liang Hu, Pedro V Sander, and Hugues Hoppe. 2010. Parallel View-Dependent Level-of-Detail Control. *IEEE Transactions on Visualization and Computer Graphics* 16, 5 (2010), 718–728.
- Peter Lindstrom and Greg Turk. 1998. Fast and memory efficient polygonal simplification. In *Proceedings of the 9th IEEE Visualization Conference (VIS '98)*.
- David Luebke, Benjamin Watson, Jonathan D Cohen, Martin Reddy, and Amitabh Varshney. 2002. *Level of Detail for 3D Graphics*. Elsevier Science Inc.
- Rafat Mantiuk, Kil Joong Kim, Allan G. Rempel, and Wolfgang Heidrich. 2011. HDR-VDP-2: A Calibrated Visual Metric for Visibility and Quality Predictions in All Luminance Conditions. *ACM Transactions on Graphics* 30, 4 (2011), 40:1–40:14.
- Matthias Nießner, Benjamin Keiwert, Matthew Fisher, Marc Stamminger, Charles Loop, and Henry Schäfer. 2016. Real-Time Rendering Techniques with Hardware Tessellation. *Computer Graphics Forum* 35, 1 (2016), 113–137.
- Jarek Rossignac and Paul Borrel. 1993. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics: Methods and Applications*. Springer Berlin Heidelberg, 455–465.
- Michaël Roy, Sebti Foufou, and Frédéric Truchetet. 2004. Mesh comparison using attribute deviation metric. *International Journal of Image and Graphics* 04, 01 (2004), 127–140.
- Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. 2001. Texture Mapping Progressive Meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, 409–416.

- Henry Schäfer, Magdalena Prus, Quirin Meyer, Jochen Süßmuth, and Marc Stamminger. 2013. Multiresolution attributes for hardware tessellated objects. *IEEE Transactions on Visualization and Computer Graphics* 19, 9 (2013), 1488–98.
- Donald Shepard. 1968. A Two-dimensional Interpolation Function for Irregularly-spaced Data. In *Proceedings of the 1968 23rd ACM National Conference (ACM '68)*. ACM, 517–524.
- Samuel Silva, Joaquim Madeira, and Beatriz Sousa Santos. 2009. PolyMeCo – An integrated environment for polygonal mesh analysis and comparison. *Computers & Graphics* 33, 2 (2009), 181–191.
- Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612. <https://doi.org/10.1109/TIP.2003.819861>
- Nathaniel Williams, David Luebke, Jonathan D. Cohen, Michael Kelley, and Brenden Schubert. 2003. Perceptually Guided Simplification of Lit, Textured Meshes. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics (I3D '03)*. ACM, 113–121.
- Andrew Willmott. 2011. Rapid Simplification of Multi-attribute Meshes. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics (HPG '11)*. ACM, 151–158.
- Yazhen Yuan, Rui Wang, Jin Huang, Yanming Jia, and Hujun Bao. 2016. Simplified and tessellated mesh for realtime high quality rendering. *Computers & Graphics* 54 (2016), 135–144.