STAG: Smart Tools & Apps for Graphics (2014)
Andrea Giachetti (Editor)

Stateless Level of Detail Lighting for Automotive Visualization

Christoph Weber[†1] and Marc Stamminger[‡1]

[1]Friedrich-Alexander University Erlangen-Nuremberg, Germany

Abstract

Car models typically consist of highly specular surfaces including sharp angles. Renderings of such models contain very fine, sharp lighting features, that both make Level-of-Detail switches well visible and result in strong aliasing artifacts. In this paper we address both issues. As Level-of-Detail approach, we apply on-the-fly vertex clustering and introduce a texture coordinate correction to remove popping artifacts, that become visible during mesh simplification. By replacing vertex normals with texture normals, we can provide stable lighting features, even if the underlying mesh changes. To prevent aliasing effects on specular highlights, we then apply a variant of LEAN mapping (linear efficient antialiased normal mapping). We observe that LEAN mapping cannot be applied directly, because in our scenario we need to filter object space *normals, as we have no presupposed tangent-space availeble. We therefore create a per-texel tangent space from an object space MIP normal, thus retaining the benefits of a 2D distribution without a preexisting tangent space. Both approaches in combination, allow us to render highly reflective, detailed models with continuous level of detail and anti-aliased lighting, at the price of moderately increased memory consumption and render time.*

1. Introduction

In this paper we look at the rendering of (rather) complex meshes on mobile devices in good quality with anti-aliasing. Efficient rendering demands Level-of-Detail (LoD) methods. Due to the little overhead in memory consumption and LoD selection we look at vertex clustering methods such as the original clustering method by Rossignac [RB93], Hoppes Progressive Meshes [Hop96] or the *POP Buffer* [LJBA13], which addresses mobile devices explicitly. Except when using very simple smooth shading, these methods require normal maps to account for the lost geometric detail. For filtering these normal maps, LEAN mapping [OB10] is a well-established method. In this paper, we address a number of issues that arise using the combination described above :

· Vertex clustering results in jittering of vertices that is also transfered to the texture, resulting in visible popping or distortion artifacts.

· LEAN mapping generates significant overhead in texture memory and requires a tangent space.

· Tangent spaces becomes discontinuous, resulting in highly visible discontinuities in normal maps.

In this paper, we show how these issues can be addressed by applying a texture coordinate correction. We introduce a modified texture-based tangent space and explain, how to significantly reduce the memory required by LEAN.

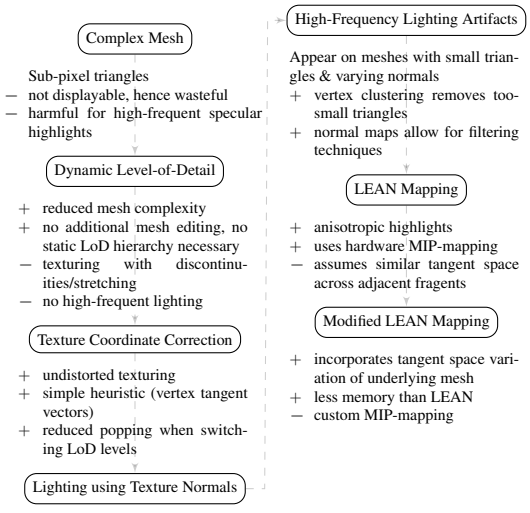

Figure 1: *Overview of our approach. We propose a texturing technique to use with vertex clustering and show how to produce artifact-free specular lighting.*

† christoph.weber@cs.fau.de
‡ marc.stamminger@informatik.uni-erlangen.de

© The Eurographics Association 2014.

DOI: 10.2312/stag.20141236

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

1.1. Related Work

Mesh Simplification. The original Progressive Meshes paper [Hop96] describes a mesh simplification algorithm based on vertex split and edge collapse operations. One implementation with the GPU by Hu et al. [HSH09] utilizes the geometry shader, which is presently not available on mobile platforms.

In contrast to progressive methods, discrete Level-of-Detail (LoD) techniques prevent dynamic changes to the mesh and provide several, increasingly coarse instances of the base mesh. Sander et al. [SM05] allow for smooth transitions between LoD instances. For further information we refer to Luebke et al. [LWC*02].

Gobbetti et al. [GMR*12] provide a view-dependent multi resolution setup, that adaptively refines a textured mesh.

Vertex Clustering and Attributes. A mesh is often parameterized with colors, normals and texture coordinates. Collapsing and simplifying a mesh without regard to the preservation of such features can severely alter the appearance of a model, so much so that discontinuities must be preserved. Hoppes' PM method [Hop96] thus simplifies the mesh only if the attributes provided change within the acceptable limitations. Sander et al. [SSGH01] construct a PM with a common texture parameterization with a uniform sampling of the texture over the mesh. The approach by Willmott [Wil11] focuses on the preservation of discontinuities and thin features with simplification during runtime. Cignoni et al. [CMR*99] preserve attribute values by resampling the original mesh.

The POP Buffer. Our proposal applies the vertex clustering technique proposed by Limper et al. [LJBA13]. They use an indexed face set where the vertex positions of a mesh are quantized using integer values. The POP-Buffer requires a preprocessing step, where the triangles and indices are reordered depending on the size of each triangle. In essence, the integer positions of each triangles three vertices are compared to one another, then their algorithm successively truncates the least-significant bit and checks if two vertices become identical. If so, the triangles are sorted into buckets, where each bucket holds the triangles that collapsed at the same level of precision. The buckets are finally appended and form the index buffer, with the coarsest triangles at the beginning and the finest at the end.

During rendering, only the first k buckets are rendered, where k is the required precision level for a specific viewing distance. All triangles with a higher level of precision, i.e. all the smaller ones, collapse and disappear during rendering and can be safely ignored. The POP-Buffer algorithm then applies a quantization of vertex positions which has the effect of a parallel vertex clustering technique.

The great benefit to this method is its ease of implementation and its minimal overhead. However, this method fails to account for mesh attributes, causing very noticeable distortions along with strong simplifications. In section 2, we propose an extension to existing vertex clustering algorithms that extrapolates texture coordinates for the simplified mesh, enabling the use of textures to convey mesh details.

Lighting. Shading and lighting significantly enhance the perception of meshes. A critical necessity for such computation are surface normals, which are usually bound to vertices. Discontinuities of surface normals are easily spotted and decrease the rendering quality. Consequently, vertex clustering techniques either preserve mesh segments with strongly varying normals or provide only low-frequency shading. As mentioned previously, we intend on using textures to convey mesh attributes. By using normal maps [PAC97, COM98] we can safely reduce mesh detail and ignore vertex attributes. However, because we consider a LoD scenario where a model is rendered at increasingly farther distances, we are faced with aliasing effects. Such effects stem from an insufficient sampling rate and are completely independent of any LoD approach discussed. As a matter of fact, highly detailed meshes with strong varying surface normals will indeed cause an aliasing effect resembling Bailey's Bead Phenomenon [Bai36] or "bead chain" [SMSS13]. The only straight-forward solutions during rendering are either to increase the sampling rate [SMSS13] or to decrease the normal variation of the mesh.

In this paper, we do not use mesh normals for shading and any triangles that cause aliasing effects are collapsed by the POP Buffer. Instead, we use textures to benefit from existing filtering techniques.

Normal Map Filtering. A Bump [Bli78] or Normal Map [PAC97, COM98] cannot be filtered linearly without smoothing out the surface detail. The effect is that rough surfaces become mirror-like, and crevices and ridges are smoothed over or even removed.

We pursue the idea brought forth by Olano et al. and reflected their LEAN Mapping [OB10] technique. They modify the Ward [War92] shading model and replace the specular term of the Blinn-Phong [Bli77] model with a Gaussian estimation. In comparison to Olano's efficient approach, which presupposes a tangent space to compute a Beckman distribution [BS63], we describe a scenario where the tangent space is distorted by mesh simplification and sub-pixel triangles. We discard a 3D Gaussian similar to the proposal of Olano et al. [ON97]. Instead, we devise a tangent space relative to the averaged texture normal and the fragment-specific tangent vectors.

The last filtering approach we reflected was devised by Toksvig [Tok05]. His method uses the length of filtered normal vectors to estimate the attenuation of the specular highlight on distant, rough surfaces. Yet, he does not provide the means to handle anisotropy as it appears along ridges and crevices. Incidentally, we want to handle those cases

© The Eurographics Association 2014.

Table 1: *Notation used throughout this paper.*

Symbol	Description
P_0, P_1, P_2	positions of triangle vertices
$[xyz]_{[012]}$	x, y and z component of vertex positions
$[uv]_{[012]}$	u and v component of vertex texture coordinates (tc)
$\Delta x, \Delta y, \Delta z$	x, y, z of triangle edges and normal
$\Delta u, \Delta v$	u, v of differential tc along edges
Δt	distance of point to triangle surface
$[\,\mathrm{T}\ \mathrm{B}\ \mathrm{N}\,]$	tangent space comprising tangent, bi-tangent and normal
$\delta_{\,uv}/\delta_{\,xyz}$	change of tc depending on change of position
$\delta_{\,xyz}/\delta_{\,uv}$	change of position depending on change of tc
H	Half-way vector
n	specularity coefficient
$cos\,\langle \mathrm{H}, \mathrm{N} \rangle$	cosine of scalar product between H and N
Σ	covariance matrix
I	identity matrix
μ_2	second moments of normals
λ_i	i'th level of MIP map ($i = 0$: finest resolution)

properly, as they are the cause of the aforementioned bead-chains.

2. Texture Coordinate Correction

Our approach uses the POP-Buffer [LJBA13] because of its very small rendering overhead (mapping of integer to float domain in the vertex shader). The proposed principles apply for any vertex clustering technique, that moves vertices and does not already re-sample the original mesh.

2.1. Vertex Clustering and Attribute Displacement

Simplifying meshes through vertex clustering is done by collapsing edges and thus reducing the number of distinct vertices. Along with this collapse, one vertex is superimposed upon another. This positional displacement, however, does not change the other vertex attributes, such as the texture coordinates. The effect can be observed in Figure 2 where the texture footprint of small triangles is stretched dramatically while simultaneously most of the original texture is not present at all. The simplest solution to the problem of shifted and distorted attributes that occur during mesh simplification is to re-sample the original mesh and to assign new attributes.

2.2. Correcting Texture Coordinates using Inverse Tangent Space

For our simplification approach we apply the POP Buffer [LJBA13] to reduce detail during rendering as is requires very little overhead compared to other progressive methods. Because the final quantization/simplification is executed in the vertex shader, we have very little to no information about the neighborhood. Assigning the correct attribute values for dislocated vertices however, requires a re-sampling of mesh attributes. This reasoning

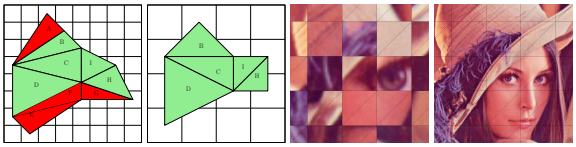

Figure 2: *From left to right: The first two images show the degeneration of triangles caused by the reduced precision of vertex positions. The third image shows a textured mesh after a severe mesh simplification. The texture coordinates migrate with the dislocated vertices and distort the final image. The fourth image shows the result of our texture coordinate correction.*

can, in essence, be summarized as a static LoD hierarchy. We propose to strike a balance by using localized information, namely the inverse of the vertex' tangent vectors, to compensate for the positional shift during quantization.

The tangent space describes the change in position based on the difference between texture coordinates and provides a parameterization relative to an objects surface (see Fig. 3 and Eq. (1)). The inverse of the tangent space describes the shift in texture coordinates, depending on the positions of the underlying mesh(see Eq. (2)). That means, any position transformed with the inverse of the tangent space is projected onto the surface spanned by the normal, i.e. into the texture space. Differences in positions are thus mapped to differential texture coordinate.

$$\left[\,\mathrm{T}\ \mathrm{B}\ \mathrm{N}\,\right] \quad \left[\,\Delta u\ \Delta v\ \Delta t\,\right] = \left[\,\Delta x\ \Delta y\ \Delta z\,\right] \equiv \frac{\delta(x,y,z)}{\delta(u,v)} \quad (1)$$

$$\left[\,\mathrm{T}\ \mathrm{B}\ \mathrm{N}\,\right]^{-1} \left[\,\Delta x\ \Delta y\ \Delta z\,\right] = \left[\,\Delta u\ \Delta v\ \Delta t\,\right] \equiv \frac{\delta(u,v)}{\delta(x,y,z)} \quad (2)$$

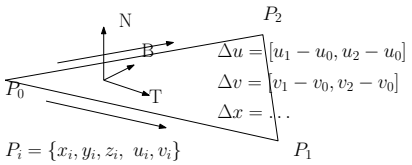

Figure 3: *Tangent space per triangle. Each vertex of a triangle is defined by its position (xyz) and its texture coordinates (uv). Using the differential positions and texture coordinates, we form the tangent space for every triangle(see Eq. (3)).*

$$\left[\,\mathrm{T}\ \mathrm{B}\ \mathrm{N}\,\right] \begin{bmatrix} u_1 - u_0 & v_1 - v_0 & 0 \\ u_2 - u_0 & v_2 - v_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 & N_x \\ y_1 - y_0 & y_2 - y_0 & N_y \\ z_1 - z_0 & z_2 - z_0 & N_z \end{bmatrix} \quad (3)$$

The correction of texture coordinates is done in the vertex shader (see Alg. 1). Here, we compute the difference between the original and the quantized position. Note that af-

© The Eurographics Association 2014.

Algorithm 1 The algorithm computes the difference between the original (e.g. the position at maximum precision) and the shifted/quantized position. The difference is multiplied with the partial derivates of the texture coordinates. The resulting differential value is added to the original texture coordinates.

```
float3   position_maxPrec , position_quant
float3x2 δuv/δxyz
float2   t_old

float3 Δposition = position_maxPrec − position_quant
float2 Δt = δuv/δxyz ∗ Δposition

float2 t_new = t_old + Δt
```

ter simplification, adjacent triangles may not share the same vertices, the only common information is the new quantized vertex position, with which we compute the positional shift. This differential position is then multiplied with the derivates of the texture coordinates (see Alg. 1) Finally, we update the texture coordinates with the calculated values. Using these corrected texture coordinates, we can collapse the mesh while preserving a plausible surface parameterization. Because the tangent space used for correction is formed by averaging the triangle tangent spaces of the vertex' neighborhood (see Fig. 3), we can expect good results for small positional deviations. This method works best for flat surfaces, and if the mesh is a plane as shown in Figure 2, we can even reconstruct the original image. In other words, because we extrapolate the texture coordinates using a linear model based on local information, we expect the method to work well for cases where the surface is similar to the local tangent space.

Note that the extrapolation can exceed the borders of textures, thus the method requires an additional margin at the borders of every texture. Such a problem, however, is not new and is often encountered when MIP mapping textures.

2.3. Using Object Space Normal Maps for Lighting

As shown in Figure 2 (third image), quantization will cause the stretching of and discontinuities in attributes. Normals should be continuous, however, especially if the lighting model uses a specular term. Taking a look back at the previous section, we showed that we can compensate for such stretching and discontinuities in textures. By replacing the mesh normals with an object space normal map, we can shade and illuminate a simplified mesh with much less of the otherwise inevitable discontinuities.

3. Modified LEAN Mapping

This section of our proposal addresses lighting of (simplified) meshes with focus on highly specular reflections. Above, we introduced a phenomenon referred to as "bead chain" [SMSS13] which occurs in crevices and on ridges where the normal changes quickly. Here, adjacent fragments receive very different normals, and without neighborhood information the only solution to avoid lighting artifacts is super sampling. An effective alternative lies in providing additional neighborhood information. By making a normal map a requirement, we can apply texture filtering instead of expensive and insufficient super sampling. One of such filtering techniques for normal maps is LEAN Mapping [OB10], it creates covariance matrices for every normal. The evaluation of the specular Blinn term is replaced with a Gaussian with the half-way vector as a sample. The ordinary specular lobe can be easily replicated (see Eq. (4)) but by providing the covariance values we provide stable specular highlights.

$$cos\,\langle \mathrm{H},\mathrm{N}\rangle^n \approx n\sqrt{|1/n\mathrm{I}|}^{-1}\, e^{-.5(\mathrm{H}\text{-}\mathrm{N})\,(1/n\mathrm{I})\,(\mathrm{H}\text{-}\mathrm{N})^T} \qquad (4)$$

Original LEAN Mapping presupposes an underlying tangent space into which the normals and half-way vector are transformed. Tangent space is either precomputed and stored per vertex or computed on the fly using pixel based derivation of positions and texture coordinates. Either way, it is not suitable to be used together with vertex clustering because of the aforementioned discontinuities of attributes. Those discontinuities are not limited to vertex clustering. Whenever adjacent triangles are smaller than pixels and feature varying normals, the respective tangent spaces vary.

3.1. 3D LEAN Mapping

A 3D Gaussian along with 3D Covariances effectively solves this problem. Note that a 3D distributions requires 6 MIP mapped covariance values in addition to 3 MIP mapped normal values. However, we show in Section 3.5, that ¾ of the covariances can be safely ignored. The actual problem when using a 3D Gaussian for specular lighting is the inversion of a 3×3 matrix during lighting. Unlike the original LEAN, which stores the first two moments and inverts a 2×2 covariance matrix on the fly, the covariances for a 3D LEAN should be precomputed, thus keeping the expensive inversion out of the lighting computations. There are two additional problems, when pre-computing and storing entire covariances. The first is the specularity term that is added to the covariance matrix and which may increase the necessary precision for storage. The other problem relates to the inversion of the covariance matrix. Because we store the inverse of the covariance matrix, we have to store the determinant of the covariance matrix as well. In short, while using 3D LEAN is clearly possible, we advocate for using a 2D Gaussian.

3.2. Texture Derived Tangent Space

By using a tangent space, the lighting computation requires only a 2D Gaussian instead of a 3D Gaussian. That means that we need to store only 3 instead of 6 distinct covariance values and use a simple 2D matrix inversion instead of a far

© The Eurographics Association 2014.

more expensive 3D one. Unlike the original LEAN mapping, our clustering approach forbids vertex based tangent vectors to create a tangent space for lighting. Furthermore, since the tangent space is derived from the underlying mesh normal, it is prone to the same problems that cause the "bead chain" artifacts described earlier, i.e., if the surface normal changes greatly within only a few fragments, the sampled normals and thus the derived tangent spaces vary as well. We propose, therefore, a modified covariance evaluation where we create a tangent space from the average normal.

3.3. Computing Covariance

The computation of covariances relative to the average normal requires two passes (see Alg. 2). We start with two textures, that contain object space normals and tangents. The tangent texture is only temporary however, and discarded after creating the covariances. During MIP map creation, we average the normal and the tangent of the input textures, for every texel in the MIP map pyramid. From these two vectors and the resulting bi-tangent, we construct the tangent space matrix. In a second pass, we compute the covariance values for each MIP texel, by iterating over the uncompressed normal map and transforming the vectors into tangent space. Then, we project each vector onto the tangent plane and compute the second moments μ_2, which we store in a 3-channel texture. Note that contrary to the original LEAN, the first moments μ_1 of our refinement are always 0 (average normal projected into modified tangent space). Contrary to our approach, original LEAN projects all normals into the same space, thus discounting variations of the underlying tangent spaces. We, on the other hand, transform the adjacent object space normals into the texel specific tangent space, which again, is the average of all underlying tangent spaces of the neighborhood, and then proceed to compute the covariance. By doing so we account for the variation of tangent spaces during covariance computation.

3.4. Specularity Computation

Our algorithm uses a modified tangent space to compute a 2D Beckmann distribution formed from the average texture normal and tangent. Yet because we discard the tangent texture, we have to compute an additional reference vector, i.e. a tangent. Considering the previously discussed topic of attribute stretching and distortion (see Fig. 2), it is evident vertex attributes are not reliable. Instead, we compute the tangent space on a per-fragment basis, by using forward/backward differences (see Alg. 3). Here, we benefit from our texture correction scheme, that we introduced in section 2. Because we interpolate/extrapolate the texture coordinates with respect to the positional shift caused by quantization, we can use the coordinates confidently for the creation of the fragment tangent space.

Once the tangent space has been compiled, we transform the half-way vector. Note that the transformed MIP normal (i.e. the expected value) is always 0.

The covariance matrix is formed by using only the second moments μ_2 and the specularity term. The final evaluation of the specular lobe is a Gaussian as described in Algorithm 3.

Algorithm 2 The covariance is computed for to every texel of the normal MIP texture. The tangent space is created using the average normal and tangents. Every normal is then transformed into the tangent space and projected onto 2D. The second moments of the 2D normal are accumulated, normalized and stored in a texture. The algorithm is executed for every MIP Level separately.

```
float3 Ñ,T̃       //average Normal \& Tangent
for(texCood : MIP level footprint )
{
  Ñ += tex2D(normalMap, texCoord);  //baked using original
  T̃ += tex2D(tangentMap, texCoord); //mesh parameter
}
float3x3 TBN = float3x3(T̃, Ñ × T̃ ,Ñ);
float3 μ2;          //second moments / covariance of Ñ
for(texCood : MIP level footprint )
{
  N = tex2D(normalMap,  texCoord);
  float3   N_tbn  = N * TBN ;
           N_tbn  = N_tbn/N_tbn.z;

  μ2 += float3( N_tbn.x², N_tbn.y², N_tbn.x * N_tbn.y );
}
μ2 /= #MIP level            //normalize
tex2D(covarianceMap) =  μ2 //store covariance
```

Algorithm 3 The average normal and covariance values are read from textures. We compute the screen space derivative of position and texture coordinate and form the tangent space using the texture normal as a reference. We transform the half-way vector into the tangent space, where the normal is always 0, and evaluate the specularity using a 2D Gaussian.

```
float3 N = tex2D(normalMap, texCoordCorrected);
float3 μ2 = tex2D(covarianceMap, texCoordCorrected);

float3 Δpx = δ position / δ x;   // screenspace derivation
float3 Δpy = δ position / δ y;
float2 Δtx = δ texCoordCorrected / δ x;
float2 Δty = δ texCoordCorrected / δ y;

float3 B = normalize(N) × normalize(Δpx*Δty.t − Δpy*Δtx.t);
float3 T = normalize(N) × B;

float3x3 TBN = float3x3(T,B,N);
float3   H_tbn  = H * TBN ;
         H_tbn  = H_tbn/H_tbn.z;

float2x2 Σ( μ2.x + 1/s, μ2.z ,
            μ2.z ,      μ2.y + 1/s );

float spec =  1/(s√|Σ|) exp(−.5 H_tbn Σ⁻¹ H'_tbn)
```

The screen-space derivation accounts for most of the complexity. A desirable option is to compute the tangent vectors in a geometry shader, such a solution however cannot create smooth vertex tangents. Furthermore, current mobile hardware does not support geometry shaders.

© The Eurographics Association 2014.

3.5. Omitting Covariances

Original LEAN mapping requires 3 channels in addition to the MIP map pyramid to store the two-dimensional covariance matrix, but we have noticed that $3/4$ of that information is useless. Because the lowest level is the covariance of the expected value itself, i.e. $\Sigma = 0$. Assuming that $\lambda = 0$ denotes the largest and finest layer, we skip the lowest MIP map level and store only the layers starting at $\lambda = 1$.

While computing lighting, we force the application to access a finer level than it would based on the automatic calculation. We then evaluate the MIP map level λ, if $0 \leq \lambda < 1$ we scale the covariance Σ_{λ_1} using the value λ, thus interpolating between $\Sigma_0 = 0$ and the first layer in the covariance MIP map Σ_{λ_1} (see Alg. 4). We must take special care, when baking the specularity exponent s into the texture, in that case the lowest covariance values are not 0 but $1/s\,\mathrm{I}$, with I being the identity matrix. Note that this omission is particular effective when using a 3D Gaussian including 6 covariances and one determinant.

In order to achieve smoother results, we recommend to use the covariance sooner than later, i.e. blurring the high-light at a closer distance.

Algorithm 4 When omitting the lowest level of the covariance, we have to force the application to interpret the lowest level as the first. The covariance, in our case the second moments μ_2, should then be scaled with the MIP map level λ to allow sharp specular features at high magnification and close distance.

```
float3 N = tex2D(normalMap, texCoordCorrected);
float3 μ2 = tex2D(covarianceMap, texCoordCorrected, λ − 1);

if(0 ≤ λ < 1)
        μ2 *= λ;
... // continue as before
```

4. Results

4.1. Texture Coordinate Correction

Figure 2 in Section 2.1 illustrates the effectiveness of linear extrapolation when correcting the texture coordinates of displaced vertices. Here, we show that even very non-planar meshes profit from our correction scheme. In Figure 4 we display a textured sphere. To visualize the impact of our method, we depict the spheres in middle and the right images, using strong vertex clustering by quanizing vertex positions to 3 bits. The middle images show the results of vertex clustering without texture coordinate correction, here the texture is extremely distorted so that checkerboard pattern is lost. With texture coordinate correction enabled, the texture is still well reproduced, Yet, there are still very striking artifacts, particularly the discontinuities between adjacent faces. However, without global information or re-sampling the original parameterization, we cannot reconstruct a continuous appearance. In order to show the efficacy of our correction when surface shading, we provide a rendering of a sphere with environment map. Some detail, such as the sky, is hidden by the geometry and the aforementioned discontinuities are also present. However, we note, that such coarse quantization is likely to appear at a distance were the textures are blurred by MIP-mapping. As shown in the accompanying video, our texture coordinate correction also results in less visible popping in lighting when switching between LoD levels.

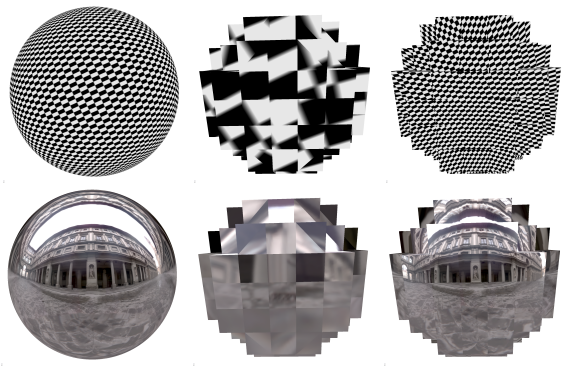

Figure 4: *Top row: checker board texture. Bottom row: Environment map using object space normal map. Left: original sphere. Center: strong quantization (3 bit) without texture coordinate correction. Right: same quantization as center, corrected texture lookup in normal map (vertex normal is useless). Note the discontinuities between faces that show the limitations of an estimation without global information.*

4.2. LEAN Mapping

In figure 5 we demonstrate our LEAN mapping to solve the problem of the "bead-chains" that appear at the ridges of the door frame and at the creases of the doors themselves. For the upper images we have used vertex normals and blinn-phong shading. The lower images were generated using vertex quantization and LEAN mapping. The images on the right show a 5.5× magnification of the model at a far distance (hence the distortion of the body). The upper right model is rendered using 30k vertices whereas the model on the bottom right is depicted using only 18k triangles. Note that the "bead-chain" artifact already appears in the near distance rendering of the upper left image (high-light on the door frame).

LEAN mapping requires an additional MIP mapped 3 channel texture besides the MIP mapped normal map to provide covariances, thus demanding $8/3$ the memory of the mere normal map. If the covariance requires twice the precision of the normal, LEAN will demand four times the memory of the normal map. However, by omitting the largest layer

© The Eurographics Association 2014.

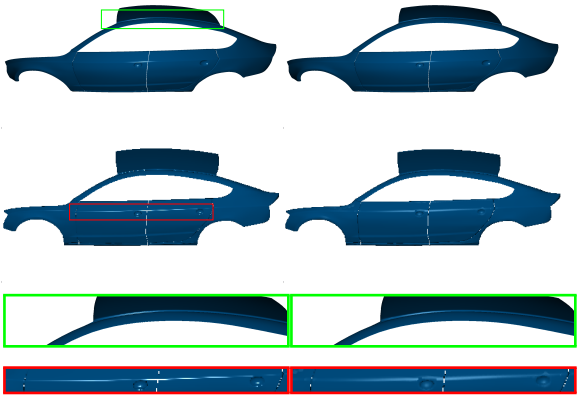

Figure 5: *Rendering of a car body. Left column: Shading with Vertex Normals. Right column: Shading with LEAN Mapping. Top row: Near distance, notice the jagged highlight in the upper left image, between roof and front door. Bottom row: Far distance at 5.5× magnification.*

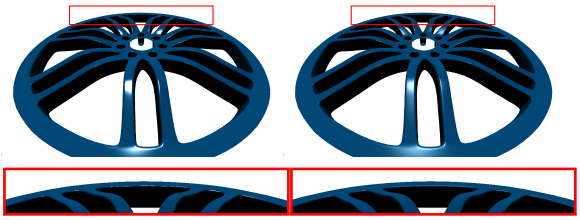

Figure 6: *Rendering of a rim. With zoom to the critical regions. Note the jagged highlights at the edge on the left.*

of the covariance texture (See Section 3.5.), we only require an additional ⅓ of the initial normal map (given equal precision, see Fig. 7).

The bottleneck of our application is the computation of the tangent space, specifically the screen space derivation of position and texture coordinates. Note that vertex attributes cannot be used in conjunction with simple vertex clustering. In comparison, when employing vertex attributes, we must ensure that adjacent fragments use similar tangent and normal vectors, i.e. that the underlying mesh is always coarse enough to eliminate the "bead-chain" phenomenon.

Table 2 shows the timings of the renderings show in figure 5 at 1920×1080 pixels. We experiment on two GPUs: an Adreno 330(Snapdragon 800) and a desktop NVIDIA 660 GTX. We render using a) the vertex normal, b) the texture normal and c) the texture normal and covariance. Note that the near-field rendering puts far more strain on the fragment computation than the rendering of the distant object (approx 5× surface area).

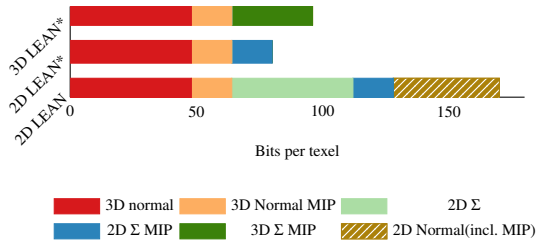

Figure 7: *Required bits per texel for different implementations of LEAN. '*' indicates the omission of λ_0 covariances. Note that we assume 16 bit precision, also note that the original LEAN stores an additional 2D Normal including MIP hierarchy, which could be computed on the fly.*

Table 2: *Timings in ms, for renderings of the car body at 1920×1080 pixels. Measurements were taken at a far, and a near distance, using either vertex normal, texture normal, or LEAN mapping as described in Algorithms 3 and 4.*

	NVIDIA 660 GTX			Adreno 330		
	Vert. N.	**Tex. N.**	**LEAN**	**Vert. N.**	**Tex. N.**	**LEAN**
Near	0.33	0.40	0.55	8.68	8.76	12.83
Far	0.13	0.17	0.17	2.98	3.72	4.66

5. Limitations

The texture coordinate correction uses the inverse of each vertex' tangent space to compensate for the positional shift during vertex clustering. Lacking global information, we cannot perfectly reconstruct the original parameterization. Furthermore, the degree of quantization is limited by the parameteization, too large shifts may cause texture cordinates to exceed the borders of textures.

6. Conclusion

We have employed a stateless Level-of-Detail lighting approach, effectively removing aliasing on specular highlights. Our first contribution is a texture coordinate correction to remove stretching and distortion of textures that is typical for mesh simplification via vertex clustering. By storing the mesh normals in textures, we can us specular lighting as well as environment mapping along with vertex clustering, and without noticeable discontinuities. In addition we employ a modified LEAN mapping to remove the "bead-chain" artifacts, which appear where normals change quickly, e.g. on ridges and creases. Our second contribution hence addresses the creation and the use of tangent space, as we provide covariances relative to the MIP normal, thus accounting for the tangent space variation across the mesh. Finally, we recognize the fact that most of the covariance data used for specular shading is useless and can be removed easily.

© The Eurographics Association 2014.

Acknowledgements

This work was partly supported by the Research Training Group 1773 "Heterogeneous Image Systems", funded by the German Research Foundation (DFG).

References

[Bai36] BAILY F.: On a remarkable phenomenon that occurs in total and annular eclipses of the sun. *Monthly Notices of the Royal Astronomical Society 4* (1836), 15. 2

[Bli77] BLINN J. F.: Models of Light Reflection for Computer Synthesized Pictures. *SIGGRAPH Comput. Graph. 11*, 2 (July 1977), 192–198. 2

[Bli78] BLINN J. F.: Simulation of Wrinkled Surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (Aug. 1978), 286–292. 2

[BS63] BECKMANN P., SPIZZICHINO A.: *The scattering of electromagnetic waves from rough surfaces*. International series of monographs on electromagnetic waves. Pergamon Press; [distributed in the Western Hemisphere by Macmillan, New York], 1963. 2

[CMR*99] CIGNONI P., MONTANI C., ROCCHINI C., SCOPIGNO R., TARINI M.: Preserving attribute values on simplified meshes by resampling detail textures. *The Visual Computer 15*, 10 (1999), 519–539. 2

[COM98] COHEN J., OLANO M., MANOCHA D.: Appearance-preserving Simplification. In *Proc. of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 115–122. 2

[GMR*12] GOBBETTI E., MARTON F., RODRIGUEZ M. B., GANOVELLI F., DI BENEDETTO M.: Adaptive Quad Patches: An Adaptive Regular Structure for Web Distribution and Adaptive Rendering of 3D Models. In *Proc. of the 17th International Conference on 3D Web Technology* (2012), ACM, pp. 9–16. 2

[Hop96] HOPPE H.: Progressive Meshes. In *Proc. of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 99–108. 1, 2

[HSH09] HU L., SANDER P. V., HOPPE H.: Parallel View-dependent Refinement of Progressive Meshes. In *Proc. of the Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 169–176. 2

[LJBA13] LIMPER M., JUNG Y., BEHR J., ALEXA M.: The POP Buffer: Rapid Progressive Clustering by Geometry Quantization. *Comput. Graph. Forum 32*, 7 (2013), 197–206. 1, 2, 3

[LWC*02] LUEBKE D., WATSON B., COHEN J. D., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002. 2

[OB10] OLANO M., BAKER D.: LEAN Mapping. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D '10, ACM, pp. 181–188. 1, 2, 4

[ON97] OLANO M., NORTH M.: *Normal distribution mapping*. Tech. rep., 1997. 2

[PAC97] PEERCY M., AIREY J., CABRAL B.: Efficient Bump Mapping Hardware. In *Proc of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 303–306. 2

[RB93] ROSSIGNAC J., BORREL P.: Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics* (1993), Falcidieno B., Kunii T. L., (Eds.), IFIP Series on Computer Graphics, Springer, pp. 455–465. 1

[SM05] SANDER P. V., MITCHELL J. L.: Progressive Buffers: View-dependent Geometry and Texture LOD Rendering. In *Proc. of the Third Eurographics Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2005), SGP '05, Eurographics Association. 2

[SMSS13] SIEGL C., MEYER Q., SUÍNER G., STAMMINGER M.: Technical Section: Solving Aliasing from Shading with Selective Shader Supersampling. *Comput. Graph. 37*, 8 (Dec. 2013), 955–962. 2, 4

[SSGH01] SANDER P. V., SNYDER J., GORTLER S. J., HOPPE H.: Texture Mapping Progressive Meshes. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 409–416. 2

[Tok05] TOKSVIG M.: Mipmapping Normal Maps. *J. Graphics Tools 10*, 3 (2005), 65–71. 2

[War92] WARD G. J.: Measuring and Modeling Anisotropic Reflection. *SIGGRAPH Comput. Graph. 26*, 2 (July 1992), 265–272. 2

[Wil11] WILLMOTT A.: Rapid Simplification of Multi-Attribute Meshes. In *High-Performance Graphics* (August 2011). 2

© The Eurographics Association 2014.