Photic Extremum Lines

Markus Pawellek
markus.pawellek@mailbox.org

Abstract

In the field of illustrative visualization, feature lines are essential for conveying the shape of a given object. Photic extremum lines (PELs) are a type of feature lines in object space, which are, besides surface geometry and view position, dependent on the illumination. In this way, illustrations generated by PELs strongly coincide with line drawings created by hand due to human perception. Furthermore, PELs are easily adjustable by switching between lighting models and allow various post-processing techniques for line stylization and shading. The algorithm to extract PELs from scenes mainly has to compute up to third-order derivatives for each vertex and may be parallelized. Implementations allowing real-time performance exist for the CPU and GPU. By comparison to other feature line types, it can be seen that PELs are an effective and flexible tool for scientific illustration.

Keywords: *Illustrative Visualization, Non-Photorealistic Rendering, Feature Lines, Object-Space Algorithm, Contours, Silhouettes, Suggestive Contours, Photic Extremum Lines, Illumination*

1 Introduction

Illustrative visualization is the science and art of effectively communicating known aspects of scientific data in an accurate and intuitive way. Especially for the rendering of volumetric data sets in medicine, it is a valuable tool to reduce a vast amount of complex information to its essence. In this respect, photorealistic rendering techniques are suboptimal because they are not able to efficiently depict features of interest. Our knowledge of human cognition shows that, artistic drawings or paintings, in comparison to a photograph of the same scene, seem to be more suitable for communication and more pleasing in visual experience (Xie et al. 2007). Therefore non-photorealistic rendering techniques, typically inspired by artistic styles, are used to create such illustrations. (Viola et al. 2005)[1]

Feature lines represent a given data set as a line drawing to mimic hand-drawn illustrations. In such a way, a large amount of information can be communicated in a succinct manner by taking advantage of human visual acuity. Used as an abstraction tool in illustrative visualization, feature lines convey the shape of objects much more efficiently compared to a photograph. (Isenberg et al. 2003; Viola et al. 2005; Xie et al. 2007)

There are many different types of commonly-used feature lines, such as contours (Isenberg et al. 2003), suggestive contours (DeCarlo et al. 2003), ridge-valley lines (Ohtake, Belyaev, and Seidel 2004), apparent ridges (Judd, Durand, and Adelson 2007), and demar-

[1] In this report, citations concerning more than one sentence are given at the end of the respective paragraph.

cating curves (Kolomenkin, Shimshoni, and Tal 2008). Typically, these only depend on the surface geometry, such as normal and curvature, and possibly the view position. However, human perception is highly sensitive to high variations in illumination. As a consequence, for conveying the shape of objects according to human perception, feature lines should also depend on the lighting of an object. (Xie et al. 2007; Zhang et al. 2011)

In this report, we present the concept and implementation of photic extremum lines (PELs), one of the first types of feature lines exhibiting a dependency on illumination. PELs have been first introduced in Xie et al. (2007) and further developed in Zhang, He, and Seah (2010). Strongly inspired by the edge detection techniques for 2D images, they are characterized by a sudden change of illumination on the surface of a 3D object. Since their computation is taken out in object space, PELs are flexible and enable further post-processing such as line stylization and shading (Isenberg et al. 2003). Furthermore, by manipulating the illumination of an object, the user can take full control to adjust the rendering output and achieve desired illustration results. Implementations for PELs can be done for the CPU and GPU, nowadays, achieving real-time performance. (Xie et al. 2007; Zhang, He, and Seah 2010)

2 Related Work

For the comprehension and implementation of PELs, we also need to rely on several basic techniques and definitions. In Isenberg et al. (2003), we get a thorough classification of feature line types together with recommendations according to the requirements of an application. It also describes the general routine for feature line extraction by using subpolygon interpolation which is also used in Zhang, He, and Seah (2010) to render PELs. Furthermore, a basic but general approach for the hidden line removal for object-space algorithms by using a two-pass rendering with depth buffer testing is provided. And additionally, using the above techniques the algorithm for extracting the contours of an object is explained. (Isenberg et al. 2003)

The definition of PELs involves up to third-order derivatives of scalar illumination functions given on a triangle mesh. Algorithms to correctly estimate curvatures and such derivatives are given in Rusinkiewicz (2004). The main content of this report is based on Xie et al. (2007) which defines PELs and provides a first algorithm and a whole framework to properly generate them. Supposable, the algorithm was implemented using the CPU. In Zhang, He, and Seah (2010), an improved real-time implementation for PELs on the GPU using the standard graphics pipeline for gradient computations is described. Hereby, the authors have used a simpler threshold test and a transformed equation to estimate derivatives for vertices.

3 Mathematical Preliminaries

To systematically comprehend the definition, design, and implementation of PELs, basic knowledge in differential geometry is administrable. In the following, we will repeat the definitions and formulas for gradients and directional derivatives of functions defined on a two-dimensional surface.

For this section, let S be smooth two-dimensional surface patch and $x \in S$ be an arbitrary point. We assume $u, v \in \mathrm{T}_x(S) \subset \mathbb{R}^3$ to be vectors that span the tangent space of S at the point x. Furthermore, let $f, g \colon S \to \mathbb{R}$ be two continuously differentiable scalar functions on S. To be able to express the gradient of such functions, we need the first fundamental form.

DEFINITION: (First Fundamental Form)

The first fundamental form $\mathrm{I}_{uv}(x)$ *of* S *at* x *in* uv*-coordinate representation is given by*

$$\mathrm{I}_{uv}(x) \coloneqq \begin{pmatrix} \|u\|^2 & \langle u \,|\, v \rangle \\ \langle u \,|\, v \rangle & \|v\|^2 \end{pmatrix} .$$

As direct result, its inverse can be computed in the following way.

$$\mathrm{I}_{uv}^{-1}(x) = \frac{\operatorname{adj} \mathrm{I}_{uv}(x)}{\det \mathrm{I}_{uv}(x)}$$

$$\operatorname{adj} \mathrm{I}_{uv}(x) = \begin{pmatrix} \|v\|^2 & -\langle u \,|\, v \rangle \\ -\langle u \,|\, v \rangle & \|u\|^2 \end{pmatrix}$$

$$\det \mathrm{I}_{uv}(x) = \|u\|^2 \|v\|^2 - |\langle u \,|\, v \rangle|^2$$

By using $[r]_{uv} = \mathrm{I}_{uv}(x)\, r$, these expressions allow us to freely transform the uv-coordinate representation $[r]_{uv} \in \mathbb{R}^2$ of tangent space vectors $r \in \mathrm{T}_x(S)$ back and forth to their standard representation.

DEFINITION: (Directional Derivative)

Let $w \in \mathrm{T}_x(S)$ be an arbitrary direction. Then the directional derivative $\partial_w f(x)$ of f at point x in direction w is given as follows.

$$\partial_w f(x) = \langle \nabla f(x) \,|\, w \rangle$$

This definition was provided for the sake of completion. Using the definition of directional derivatives together with the expressions based on the first fundamental form, the computation of the gradient of a given scalar function on the surface is then straightforward.

DEFINITION: (Gradient)

Let $[\nabla f(x)]_{uv}$ be the uv-coordinate representation of $\nabla f(x)$. Then the following holds.

$$[\nabla f(x)]_{uv} = \mathrm{I}_{uv}^{-1}(x) \begin{pmatrix} \partial_u f(x) \\ \partial_v f(x) \end{pmatrix}$$
$$\nabla f(x) = \begin{pmatrix} u & v \end{pmatrix} [\nabla f(x)]_{uv}$$

The definition of PELs involves a more complicated derivative operator to precisely formulate the requirements for a local maximum in a specific direction.

DEFINITION: (PEL Operator)

The PEL operator defined as follows evaluates the directional derivative of g in the direction of the gradient of f.

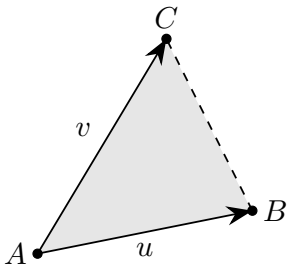
$$\mathcal{D}_f g(x) \coloneqq \left\langle \nabla g(x) \,\middle|\, \frac{\nabla f(x)}{\|\nabla f(x)\|} \right\rangle$$

For the above definition, f can be seen as a scalar field which provides a vector field of directions on S with its gradient. The function g is then differentiated at each point in a direction given by this vector field.

Example: In particular, for the interior of a triangle, characterized by the points $A, B, C \in \mathbb{R}^3$, the first fundamental form is constant and we can set u and v as follows to use the above formulas.

$$u \coloneqq B - A \qquad v \coloneqq C - A$$

To represent the function f in a discretized form for a computer, we will use a triangle mesh and will only provide a value for each vertex. For interior points x of triangles with barycentric coordinates $\alpha, \beta, \gamma \in [0, 1]$ with $\alpha + \beta + \gamma = 1$, linear interpolation over the vertex values $f(A)$, $f(B)$, and $f(C)$ is taken out.



$$f(x) = \alpha f(A) + \beta f(B) + \gamma f(C)$$

As a result, the graph of the discretized function of f is a plane for the interior of each triangle. So also its gradient is constant within the interior. Computing directional derivatives for u and v then reduces to the following expressions.

$$\partial_u f = f(B) - f(A) \qquad \partial_v f = f(C) - f(A)$$

Applying these formulas for each triangle and accumulating those values at their vertices will give us an approach to estimate the gradients and directional derivatives on triangle meshes.

4 Photic Extremum Lines

Providing the mathematical details, now, we are in a position to give a formal definition of PELs. For this section, let S again be a smooth surface patch and $\varphi \colon S \to \mathbb{R}$ a three-times continuously differentiable scalar function that describes the illumination of the surface S in grayscale values.

DEFINITION: (Photic Extremum Lines)

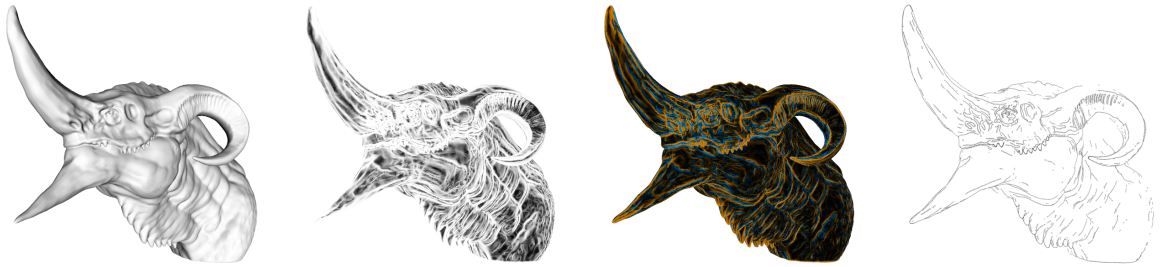
The set $\mathrm{PEL}(S, \varphi)$ of photic extremum lines consists of all points $x \in S$ such that

$$\mathcal{D}_\varphi \|\nabla \varphi\|(x) = 0 \qquad \mathcal{D}_\varphi^2 \|\nabla \varphi\|(x) < 0 \ .$$

Such points are also called photic extrema.

Let $\tau \in \mathbb{R}_0^+$ be an arbitrary threshold and $\mathcal{F}_\tau(S, \varphi)$ a filter in the following sense.

$$\mathcal{F}_\tau(S, \varphi) \coloneqq \{x \in S \mid \|\nabla \varphi\|(x) > \tau\}$$



(a) *Illumination* φ **(b)** *Variation* $\|\nabla\varphi\|$ **(c)** $\mathcal{D}_\varphi \|\nabla\varphi\|$ **(d)** $\mathrm{PEL}_\tau(S, \varphi)$

Figure 1: ***Visualization of PEL Definition***
These images show the same object with surface S. For each image, the object is shaded with a specific function or set of the PEL definition. Hereby, (c) uses orange for positive, blue for negative, and black for zero values. The images shall give an intuition on why the definition of PELs makes sense.

> *Then the set of τ-filtered photic extremum lines is defined as follows.*
>
> $$\mathrm{PEL}_\tau(S, \varphi) \coloneqq \mathrm{PEL}(S, \varphi) \cap \mathcal{F}_\tau(S, \varphi)$$

In other words, photic extrema are points $x \in S$ where the variation of illumination $\|\nabla\varphi\|$ in the direction $\nabla\varphi$ of its gradient reaches a local maximum. The definition involves the previously defined PEL operator to provide sufficient conditions for these specific local maxima. Figure 1 visualizes the different functions used in the above definition.

Our definition strongly complies with the original definition given in Xie et al. (2007). But additionally, we already added the threshold to point-wise filter out parts of lines that are too weak in the sense of light variation. This was mainly done to be mathematically rigorous in further discussions but also allows for more generalized filter techniques. The filtering is defined according to Zhang, He, and Seah (2010) to be able to achieve real-time performance. The original threshold filter method used in Xie et al. (2007) is based on a line tracing approach of Ohtake, Belyaev, and Seidel (2004) and, as a consequence, is much more involved than the point-wise approach and would most likely lead to a bottleneck in the following algorithm.

Due to the deliberately built-in dependence on illumination in the definition of PELs, we at last need to clarify which lighting model to choose. As touched in the introduction, in general, arbitrary illumination functions are allowed. The main problem is that not all known lighting models will result in near-natural line drawings. The comparison of different models would fill an article for itself and therefore is not in the scope of this report. For our purposes and according to Xie et al. (2007), we will choose a simplified Phong model without ambient or specular lighting. In this way, only diffuse lighting, that depends on the cosine of the angle between the lighting direction and the normal of the surface, is allowed. Ambient lighting can be ignored as its directional derivatives would be zero. Since specular lighting effects only arise in small areas and greatly increase illumination, they would most likely introduce unnatural and artificial lines.

Regarding the count and positioning of light sources, we again want to reduce the set of possibilities to make their specification feasible for this report. Therefore, only one light source will be used. Using this source with movements independent from the camera, PELs would not exhibit the wanted view dependency that other feature line types provide. Hence, we will position a directional light at the camera position such that light direction and view direction coincide. Due to the higher flexibility of PELs, only these restrictions make the results comparable to other feature lines not offering any kind of light dependence.

5 Algorithm

The formal definition of PELs when put together with the basic techniques of Isenberg et al. (2003) and Rusinkiewicz (2004) described in section 2 already supplies us with an intuitive approach to generate PELs. In the following, we will give the details. This time, let $\mathcal{M} = (V, E, F)$ be a regular triangle mesh consisting of vertices V, edges E, and faces F that approximates a smooth two-dimensional surface. Furthermore, let $n\colon V \to \mathbb{S}^2$ be the function that provides consistent and normalized normals for each vertex of $\mathcal{M}$. Let $S \coloneqq V \cup E \cup F$ be the surface of $\mathcal{M}$ and $\varphi\colon S \to \mathbb{R}$ a

scalar triangle mesh function representing the grayscale illumination of $\mathcal{M}$ with given values at each vertex which uses linear interpolation based on barycentric coordinates for edges and faces.

5.1 Overview

The whole PEL algorithm has to be executed each time the view position or lighting changes. This is in contrast to other feature lines that may precompute most of their needed information which are only based on surface geometry.

Concerning time and effort, the main part of the algorithm needs to compute all the illumination derivatives $\nabla\varphi$, $\mathcal{D}_\varphi \|\nabla\varphi\|$, and $\mathcal{D}^2_\varphi \|\nabla\varphi\|$ for each vertex $v \in V$. Afterwards, an iteration over all edges $[v, w] \in E$ detects photic extrema on edges. Finally, an iteration over all faces $f \in F$ connects two feature line vertices of adjacent edges if present generating feature line segments. At last, the occlusion, threshold $\tau \in \mathbb{R}^+_0$, and the value of $\|\nabla\varphi\|$ for each point decide if the respective photic extremum represented by a fragment will be rendered.

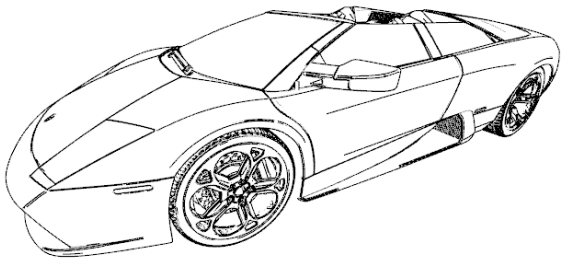
Algorithm Overview

1. Compute φ at each vertex
2. Compute $\|\nabla\varphi\|$ and $\frac{\nabla\varphi}{\|\nabla\varphi\|}$ at each vertex
3. Compute $\mathcal{D}_\varphi \|\nabla\varphi\|$ at each vertex
4. Compute $\mathcal{D}^2_\varphi \|\nabla\varphi\|$ at each vertex
5. Detect photic extrema on edges
6. Emit feature line segment for each triangle with photic extrema on adjacent edges
7. Discard fragments of feature lines based on threshold
8. Render non-occluded feature line fragments by using hidden line removal

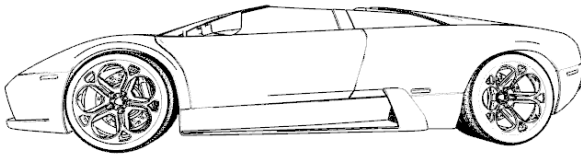
In our implementation, points 1 to 4 are taken out on the CPU. Point 5 and 6 have been merged and put inside a geometry shader of the graphics pipeline. This makes it possible to implement the threshold-based discarding of PEL fragments inside a fragment shader by a simple branch comparing the illumination variation of the fragment to the given threshold.

5.2 Preprocessing

Besides loading the triangle mesh to be rendered with PELs, we have to generate the interpolated normals for every vertex if they are not already present. For this, we refer to the standard procedures given by Max (1999) and Jin, Lewis, and West (2005). This may already be sufficient for triangle meshes that were created manually by graphic designers. Usually, such models provide us with very smooth surface normals. An example can be seen in figure 2. The car has been manually modeled and provides very smooth normals. And as a result, it shows a near-perfect feature line extraction.



(a) *Front*

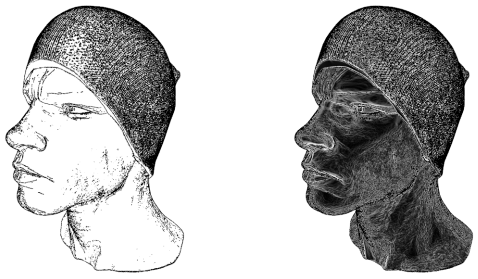


(b) *Side*

Figure 2: ***Nearly Perfect PEL Extraction for Smooth Normals*** *The images show different perspectives of the extracted PELs for the Lamborghini model that provides very smooth surface normals. It can be seen that the smoothness allows for a nearly perfect line extraction without artifacts.*

For triangle meshes originated from 3D scanners on the other hand, generated normals may be inaccurate due to errors in measurement or round-off errors. Such inconsistencies are typically called noise and would probably result in artifacts when rendering feature lines because derivative computations have a major dependency on surface normals. The effects can be seen in figure 3. As a consequence, further preprocessing would involve reducing the noise of such normals by using bilateral normal filtering for example as it was

done in Xie et al. (2007). Here, we will not provide any details concerning this topic because such filters need parameters that may be tweaked by the user.



(a) *Contours and PELs* **(b)** *Variation*

Figure 3: ***Erroneous PEL Extraction for Noisy Normals***
The images show extracted PELs and light variation of the head model. This model provides very noisy surface normals which manifest in PEL artifacts around the chin and neck. The light variation visualizes the noise of the surface normals.

For the computation of gradients and directional derivatives on the triangle mesh, each vertex should be equipped with an orthonormal system of two vectors spanning its tangent space. Generating such tangent vectors can be quite arbitrary. Our approach first uses a uniform distribution on the unit sphere to randomly generate normalized directions for each vertex. In conjunction with the vertex normal, we can then compute these tangent space vectors by two cross products.

Rusinkiewicz (2004), Xie et al. (2007), and Zhang, He, and Seah (2010) all refer to Meyer et al. (2001) when it comes to the accumulation step of face gradients at their vertices. Hereby, each gradient or directional derivative is weighted based on the Voronoi area of the point inside the current triangle with respect the whole Voronoi area of the vertex. As a consequence, these Voronoi weights should also be precomputed and stored with respect to their vertices and faces.

5.3 Gradient Computation

As said in section 5.1, the main part of the algorithm needs to compute all the illumination derivatives. All of these can be computed with the same approach.

For each face, compute the constant gradient by using formula in background. Transform and rotate this gradient into the local vertex system and accumulate multiplied with Voronoi weight. Afterwards iterate over all vertices for normalization.

Gradient Algorithm

1. For each face, compute gradient in uv-coordinate representation
2. For each adjacent vertex, transform representation into local uv-coordinates
3. for directional derivatives, compute them at vertices
4. Accumulate this representation by using Voronoi weights

5.4 Line Extraction

In section 5.1, an overview of the PEL algorithm was given. Here, the details for an implementation of points 5 and 6 are provided. The subpolygon feature line extraction method is a general method for object-space feature line algorithms described in Isenberg et al. (2003) and Xie et al. (2007). The following snippet summarizes the main points. Hereby, we define the following short notation.

$$h(x) \coloneqq \mathcal{D}_{\varphi} \left\| \nabla \varphi \right\| (x)$$

PEL Extraction

1. For each edge $[v, w] \in E$:
   (a) Check for zero-crossing:
   $$h(v)h(w) < 0$$
   (b) If this is the case, approximate zero-crossing:
   $$p \coloneqq \frac{|h(w)|\, v + |h(v)|\, w}{|h(v)| + |h(w)|}$$
   (c) Check maximum condition:
   $$\mathcal{D}^2_{\varphi} \left\| \nabla \varphi \right\| (p) < 0$$
   (d) If this is the case, emit PEL vertex p
2. For each triangle:
   (a) Check if two adjacent edges provide PEL vertices p and q
   (b) If this is the case, emit PEL segment $[p, q]$

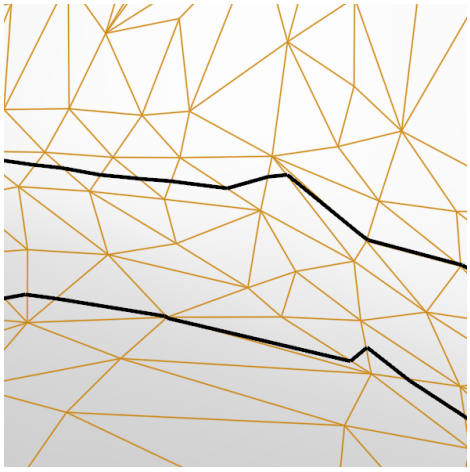


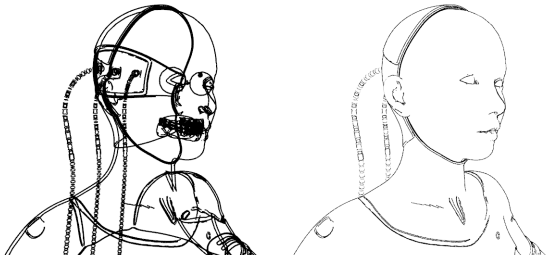
Figure 4: ***Sub-Polygon Feature Lines***
The image shows a close-up view of the mesh grid and PELs of a shaded triangle mesh. Each triangle whose adjacent edges exhibit PEL vertices show a straight line segment. As a result, neighboring triangles will connect line segments to a whole line.

A close-up view of a result of this algorithm for a given triangle mesh is given in figure 4. Here, every triangle whose edges exhibit a PEL vertex show a straight line segment connecting those vertices. Because neighboring triangles will compute the same PEL vertices on identical edges, in the overall result line segments are connected to whole feature lines.

As already described, the implementation of both points can be done in one single step by using the geometry shader of the graphics pipeline. For that matter, the input consists of triangles. For all edges of the given triangle, we will then do the steps provided in point 1 of the above algorithm. The geometry shader allows us to easily emit and render line segments. So each PEL vertex is at most computed twice but in a highly parallel approach on the GPU. This geometry shader implementation strongly aligns to the ideas given in Zhang, He, and Seah (2010).

5.5 Hidden Line Removal

For themselves, feature line algorithms that work in object space in the first instance cannot decide which lines are occluded by the object's geometry since they have no concept of depth. Not removing occluded lines results in images like the one given in figure 5. There, we also see the correct variant where only non-occluded lines are shown. According to Isenberg et al. (2003), this problem is typically referenced as *hidden line removal* and can be dealt with by using several different techniques.



(a) *Off* **(b)** *On*

Figure 5: ***Two-Pass Rendering for Hidden Line Removal***
These images show the PELs of the same model. For the left image, a two-pass rendering approach to remove by the object's geometry occluded lines has not been used. For the image on the right, first, the whole uncolored surface was rendered to write into the depth buffer and afterwards the feature lines were drawn.

Probably the simplest method involves using the depth buffer together with depth testing of the standard graphics pipeline. Hereby, the object will first be rendered as a whole but possibly uncolored surface. This makes sure that every fragment that can be seen from the camera stores its depth value into the depth buffer. All successive rendering calls will then test the depth values of their fragments against the depth buffer. In a second rendering pass, the extraction of feature lines will generate fragments with respective depth values. The built-in depth test will then discard such fragments that are occluded because their depth values are too high.

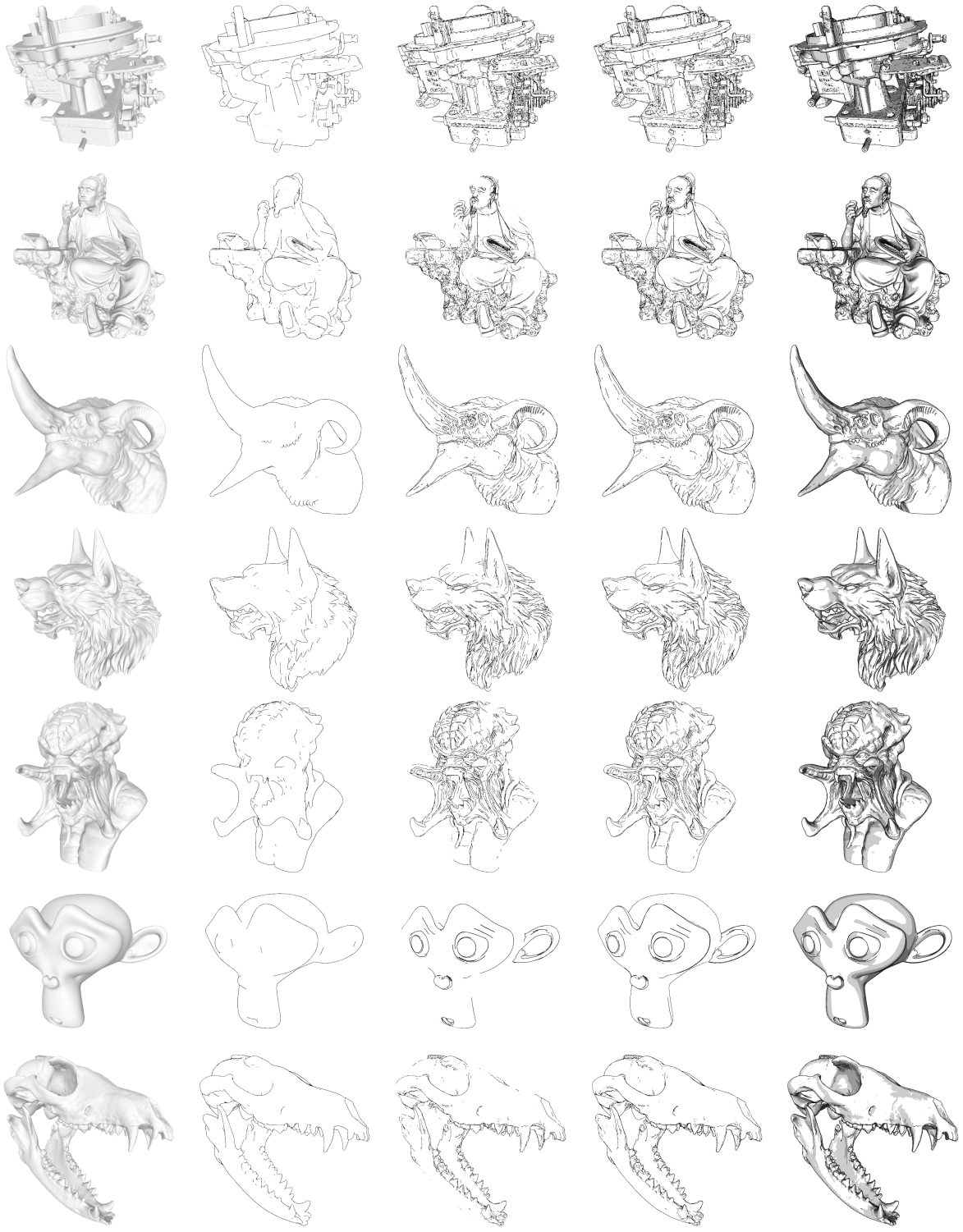
Problems?

6 Evaluation and Results

Following the definition and implementation of PELs, it is now possible to evaluate their effectiveness with respect to the illustration quality and their performance.

6.1 Quality

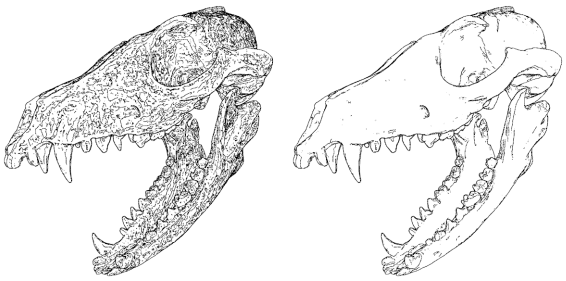
To test PELs regarding their quality, we refer to the illustrations in figure 6 where several models have been rendered with different shading techniques to make an evaluation of the effectiveness of PELs easier. We see the standard shading based on the illumination function, the objects rendered with Contours and/or PELs, and additionally applied toon shader to show future application possibilities.



(a) *Illumination* **(b)** *Contours* **(c)** *PELs* **(d)** *Contours and PELs* **(e)** *Toon Shader*

Figure 6: ***Quality Results***

The images show the carburetor, lu yu, dragon head, werewolf, predator, suzanne, and fox skull models with varying shading models. The toon shading on the right is always done in conjunction with contours and PELs. Each model provides a user-define custom threshold to filter out unwanted PELs and to maximize quality.



(a) *Low* **(b)** *Mid*

Figure 7: ***PELs for Varying Thresholds***
The images show the PELs of the fox skull model filtered with different thresholds. For the left image, a threshold near zero has been used. It can be seen that a lot of PELs shown in this image are very short and do not effectively abstract the surface information.

Contours are very robust when it comes to their extraction. Even for models with a low number of triangles, their generation algorithm creates feature lines which best describe the overall shape of the object with the strongest fore- to background separation without any measurable performance penalty. However, contours suffer from a low amount of details in inner object parts. PELs on the other hand are able to provide rich details in such areas. Compared to hand-drawn illustration, PELs seem to be consistent and natural. The main problem for PELs seems to be to robustly represent the contours of an object because light variation may not be that high at bending surfaces. But it can be seen that combining contours with PELs, in general, we receive superior results concerning the quality of illustrations.

As an object-space feature line type, PELs enable further post-processing to alter the style of lines or to apply additional shaders, such as toon shading, to the object's surface. Figure 6 applies to the already enabled contours and PELs a simple toon shading approach. Dependent on the application, this may also greatly improve the effectiveness of communicating surface features.

Taking a look at the use of user-defined thresholds for filtering unwanted PELs, figure 7 shows an example. A near-zero threshold will likely extract PELs that are very short and do not carry essential information about the surface geometry. This behavior has also been noticed for all models that are shown.

Regarding the PEL extraction, problems that arise, such as noisy lines, artifacts, or important but missing lines, may be traced back to the used lighting model to generate the illumination function. An illumination that is only based on the scalar product between view direction and surface normal could be too smooth for important large-scale details to be shown. On the other hand, small-scale unwanted noisy details are likely to be captured. To find an optimal lighting approach, we need to do further research.

6.2 Performance

As said in section 5.1, PELs have to be regenerated each time the view or lighting changes. Assuming that both are changing all the times, we have carried out some frame time measurements with our own naive implementation. The results can be seen in table 1. Hereby, we achieve real-time performance for up to hundreds of thousands of triangles. And even for up to two million triangles, the implementation provides us with interactive frame rates. As a remark, we would like to note that, when not trying to constantly animate objects but only rendering those in a viewer tool, changes in illumination and view are mainly triggered by the user and are therefore very rare. In this special case, the amortized frame time again becomes real-time capable.

Table 1: ***Performance Results***
For the listed models, this table shows the number of triangles $\triangle$, the frame time Δt, and the number of frames per second (FPS). The results were obtained by continuously extracting and rendering PELs from those objects. Each PEL extraction is defined as a frame. This benchmark was executed on a workstation with an Intel Core i7-7700K CPU and an NVIDIA GeForce GTX 1070 GPU.

Model	$\triangle$	$\Delta t\ [ms]$	**FPS**
Carburetor	751,340	158.00	6.3
Lu Yu	1,984,640	201.00	5.0
Dragon Head	941,892	215.00	4.6
Werewolf	361,629	60.50	16.5
Predator	986,719	235.00	4.2
Suzanne	15,744	0.77	1305.0
Fox Skull	699,976	41.00	24.3

7 Conclusions and Future Work

We have introduced the concept of PELs — a view- and light-dependent feature line type in object space. PELs are the set of points on an illuminated surface where the variation of illumination is larger than a given threshold and where it reaches a local maximum in the direction of its gradient. This definition stems

from the generalization of edge detection techniques for two-dimensional images to three-dimensional shapes. Hereby, our knowledge of human perception has been applied to provide an object-space algorithm that is able to generate more natural renderings of lines. We were able to see that PELs strongly coincide with hand-drawn illustrations. By definition, PELs are therefore more general than other feature line types. Changing the lighting model allows to remove unwanted details or amplify parts which would otherwise be unseen. Furthermore, applying further techniques, such as contours and toon shading, greatly improves the resulting illustration.

To extract PELs from a given scene, we have to evaluate up to third-order derivatives of a given scalar illumination function for each animation frame. This makes PELs computationally much more expensive than other feature line types. However, using today's CPUs and GPUs, real-time capability can be achieved even by naive implementations. Apart from that, all typical feature line algorithms for hidden line removal or line extraction can be used.

We have implemented PELs in C++ with an OpenGL pipeline, using a hybrid approach utilizing CPU and GPU at the same time for simplicity. The shown implementation achieves interactive up to real-time frame rates even for millions of triangles. Due to synchronization issues, computations of gradients and directional derivatives are done on the CPU in a straightforward serial algorithm. The rendering step is done in a two-pass approach to remove hidden lines. For the line extraction, we have used a geometry shader written in GLSL that outputs subpolygon feature line segments in a triangle. Fragments that do not meet the threshold are discarded by the fragment shader.

An important preprocessing step for some models should be the denoising of vertex normals by applying a bilateral normal filter. Without appropriate smoothing, a lot of non-intuitive artifact lines originate in the illustration. However, typical normal filtering uses parameters that are tweaked by the user to achieve superior results. Such adjustments quickly become infeasible for larger more complex scenes. Therefore future research and implementations should also strive for automatic normal filtering techniques without the need of user intervention as it was already shown to work in Zhang et al. (2011) for Laplacian lines. In the same category falls the automatic determination of thresholds to remove unwanted lines in a model.

Regarding our implementation, future versions should strive for a full GPU implementation according to Zhang, He, and Seah (2010). But instead of textures, a more modern approach based on compute shaders that take advantage of shader storage buffers to access information of neighboring vertices could be used. A pure GPU implementation is likely to be faster and also more flexible and easier to integrate with other techniques.

Trying to improve the quality of generated illustrations, Zhang, He, and Seah used mean-curvature lighting (Kindlmann et al. 2003; Kolomenkin, Shimshoni, and Tal 2008) instead of a simplified Phong model to make line extraction more robust. According to this, also other lighting techniques, such as exaggerated shading (Rusinkiewicz, Burns, and DeCarlo 2006), seem to be promising alternatives for a robust automatic lighting model.

References

DeCarlo, Douglas et al. (July 2003). "Suggestive Contours for Conveying Shape". In: *ACM Trans. Graph.* 22, pp. 848–855. DOI: 10.1145/1201775.882354.

Hertzmann, Aaron and Denis Zorin (2000). "Illustrating Smooth Surfaces". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. ACM Press/Addison-Wesley Publishing Co., 517–526. ISBN: 1581132085. DOI: 10.1145/344779.345074.

Isenberg, Tobias et al. (August 2003). "A Developer's Guide to Silhouette Algorithms for Polygonal Models". In: *Computer Graphics and Applications, IEEE* 23, pp. 28 –37. DOI: 10.1109/MCG.2003.1210862.

Jin, Shuangshuang, Robert Lewis, and David West (February 2005). "A Comparison of Algorithms for Vertex Normal Computation". In: *The Visual Computer* 21, pp. 71–82. DOI: 10.1007/s00371-004-0271-1.

Judd, Tilke, Frédo Durand, and Edward Adelson (July 2007). "Apparent Ridges for Line Drawing". In: *ACM Trans. Graph.* 26, p. 19. DOI: 10.1145/1276377.1276401.

Kindlmann, Gordon et al. (November 2003). "Curvature-Based Transfer Functions for Direct Volume Rendering: Methods and Applications". In: vol. 2003, pp. 513–520. ISBN: 0-7803-8120-3. DOI: 10.1109/VISUAL.2003.1250414.

Kolomenkin, Michael, Ilan Shimshoni, and Ayellet Tal (December 2008). "Demarcating Curves for Shape Illustration". In: *ACM Trans. Graph.* 27, p. 157. DOI: 10.1145/1457515.1409110.

Max, Nelson (January 1999). "Weights for Computing Vertex Normals from Facet Normals". In: *Journal of Graphics Tools* 4. DOI: 10.1080/10867651.1999.10487501.

Meyer, Mark et al. (November 2001). "Discrete Differential-Geometry Operators for Triangulated 2-Manifolds". In: *Proceedings of Visualization and Mathematics* 3. DOI: 10.1007/978-3-662-05105-4_2.

Ohtake, Yutaka, Alexander Belyaev, and Hans-Peter Seidel (August 2004). "Ridge-Valley Lines on Meshes via Implicit Surface Fitting". In: *ACM Transactions on Graphics, v.23, 609-612 (2004)* 23. DOI: 10.1145/1186562.1015768.

Rusinkiewicz, Szymon (October 2004). "Estimating Curvatures and Their Derivatives on Triangle Meshes". In: pp. 486–493. ISBN: 0-7695-2223-8. DOI: 10.1109/TDPVT.2004.1335277.

Rusinkiewicz, Szymon, Michael Burns, and Douglas DeCarlo (July 2006). "Exaggerated Shading for Depicting Shape and Detail". In: *ACM Trans. Graph.* 25, pp. 1199–1205. DOI: 10.1145/1179352.1142015.

Viola, Ivan et al. (January 2005). "Illustrative Visualization". In: p. 124. DOI: 10.1109/VIS.2005.57.

Xie, Xuexiang et al. (November 2007). "An Effective Illustrative Visualization Framework Based on Photic Extremum Lines (PELs)". In: *IEEE transactions on visualization and computer graphics* 13, pp. 1328–1335. DOI: 10.1109/TVCG.2007.70538.

Zhang, Long, Ying He, and Hock Seah (June 2010). "Real-Time Computation of Photic Extremum Lines (PELs)". In: *The Visual Computer* 26, pp. 399–407. DOI: 10.1007/s00371-010-0454-x.

Zhang, Long et al. (July 2011). "Real-Time Shape Illustration Using Laplacian Lines". In: *IEEE transactions on Visualization and Computer Graphics* 17. DOI: 10.1109/TVCG.2010.118.