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

DEFINITION 3.1: Mesh Function

$$f\colon S \to \mathbb{R}$$

DEFINITION 3.2: (First Fundamental Form Triangle)

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

$$\mathrm{I}_{uv}^{-1} = \frac{\operatorname{adj} \mathrm{I}_{uv}}{\det \mathrm{I}_{uv}} = \frac{1}{\|u\|^2 \|v\|^2 - |\langle u \,|\, v\rangle|} \begin{pmatrix} \|v\|^2 & -\langle u \,|\, v\rangle \\ -\langle u \,|\, v\rangle & \|u\|^2 \end{pmatrix}$$

DEFINITION 3.3: (Gradient Triangle)

$$[\nabla f]_{uv} = \mathrm{I}_{uv}^{-1} \begin{pmatrix} \Delta_u f \\ \Delta_v f \end{pmatrix}$$

$$\nabla f = \begin{pmatrix} u & v \end{pmatrix} [\nabla f]_{uv}$$

DEFINITION 3.4:

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

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

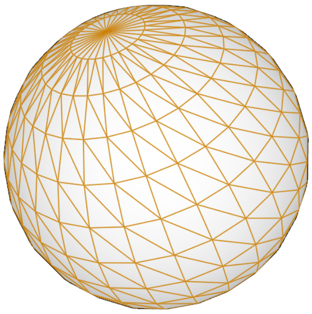


Figure 1: *Triangulated Meshes*

4 Photic Extremum Lines

For this section, let S be a smooth surface patch and $\varphi\colon S \to \mathbb{R}$ three-times continuously differentiable scalar illumination function.

In other words, photic extrema are points where the variation of illumination in the direction of its gradient reaches a local maximum.

> **DEFINITION 4.1:** (Photic Extremum Lines)
>
> *The set* $\mathrm{PEL}(S, \varphi)$ *of photic extremum lines consists of all points* $x \in S$ *such that the following holds.*
>
> $$\mathcal{D}_\varphi \left\| \nabla \varphi \right\| (x) = 0 \qquad \mathcal{D}^2_\varphi \left\| \nabla \varphi \right\| (x) < 0$$
>
> *Let* $\tau \in \mathbb{R}^+_0$ *be an arbitrary threshold.*
>
> $$\mathrm{PEL}_\tau(S, \varphi) := \{ x \in \mathrm{PEL}(S, \varphi) \mid \left\| \nabla \varphi \right\| (x) > \tau \}$$

5 Algorithm

5.1 Overview

Algorithm

1. Compute illumination
2. Compute variation
3. Compute directional derivative
4. Compute second-order derivative
5. Detect line vertices on edges by testing for photic extremum
6. Trace and filter out lines by using threshold
7. Render only visible lines by using hidden line removal

5.2 Preprocessing

Besides loading the triangle mesh to be rendered with PELs, generating interpolated normals for every vertex have to be computed if they are not already present. Here, we refer to standard procedures given by Max (1999) and Jin, Lewis, and West (2005). Further preprocessing would involve reducing the noise of such normals by using bilateral normal filter.

For the computation of gradients and directional derivatives on the mesh, local coordinate system and Voronoi weights should be precomputed. Additionally, neighboring vertices should be stored for parallelization.

5.3 Gradient Computation

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.

5.4 Line Tracing

5.5 Hidden Line Removal

6 Implementation

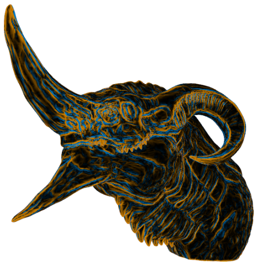
7 Results and Comparison

8 Conclusions

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

(a) *Illumination φ*

(b) *Variation $\|\nabla\varphi\|$*



(c) *Variation Slope $\mathcal{D}_\varphi \|\nabla\varphi\|$*

(d) *PELs*

Figure 2: ***Variation***
These images show the same shaded object (dragon head) to visually explain the definition of PELs.

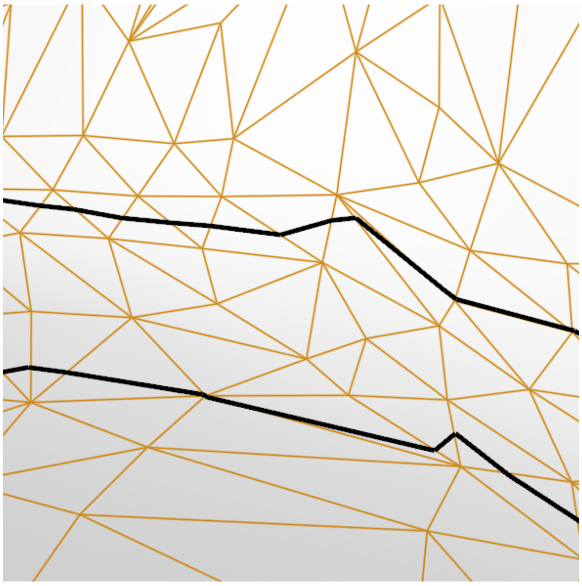


Figure 3: *Sub-Polygon Feature Lines*

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.

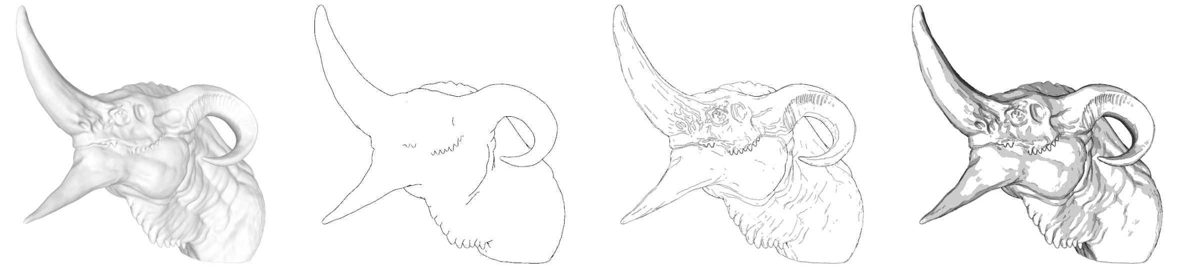
To extract PELs from a given scene, we have to evaluate up to third-order derivatives of a given scalar illumination function for each animation frame. This makes PELs computationally much more expensive than other feature line types. However, using today's CPUs and GPUs, real-time capability can be 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.

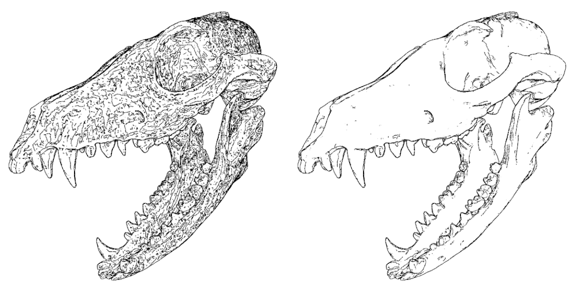
Regarding our implementation, future versions should strive for a full GPU implementation according to Zhang, He, and Seah (2010). But instead of textures, a more modern approach based on compute shaders that take advantage of shader storage buffers to access information of neighboring vertices could 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,



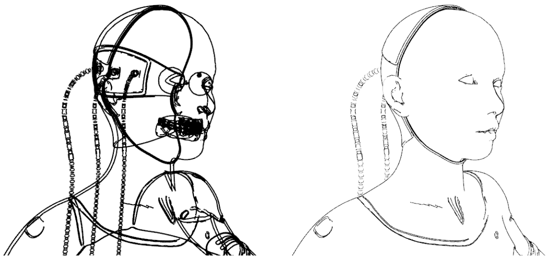
(a) *Illumination* **(b)** *Contours* **(c)** *Contours and PELs* **(d)** *Contours, PELs, and Toon*

Figure 4: ***Short Summary Part***
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



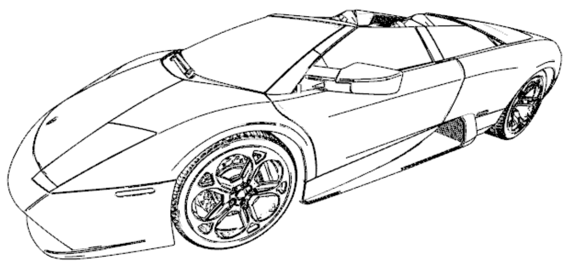
(a) *Low* **(b)** *Mid*

Figure 5: *Effect of thresholding*

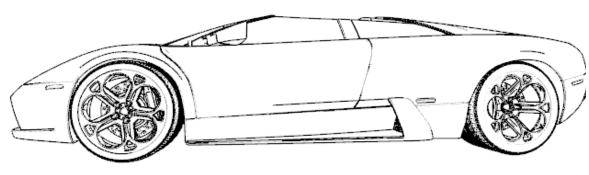


(a) *Off* **(b)** *On*

Figure 6: *Two-Pass Rendering for Hidden Line Removal*

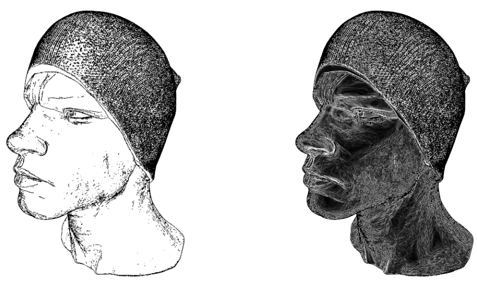


(a) *Front*



(b) *Side*

Figure 7: *Nearly Perfect Line Extraction for Smooth Objects*



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

Figure 8: *Erroneous Line Extraction for Noisy Objects*

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.