Vis Comput (2010) 26: 399–407
DOI 10.1007/s00371-010-0454-x

ORIGINAL ARTICLE

Real-time computation of photic extremum lines (PELs)

Long Zhang · Ying He · Hock-Soon Seah

Published online: 14 April 2010

Abstract Photic extremum lines (PELs) are view-dependent, object-space feature lines that characterize the significant changes of surface illumination. Though very effective for conveying 3D shapes, PELs are computationally expensive due to the heavy involvement of the third- and fourth-order derivatives. Also, they require the user to manually place a few auxiliary lights to depict the model details, which is usually tedious work. To overcome these challenges, we present a novel computational framework that improves both the speed and quality of PELs. First, we derive a simple, closed-form formula of gradient operator such that various orders of derivatives can be computed efficiently and in parallel using graphics processing units (GPUs). The GPU-based PEL extraction algorithm is one order of magnitude faster than the original one. Second, we propose to extract PELs from various non-photorealistic shadings that not only depict the overall shape but also bring out the details at different frequencies simultaneously. As a result, the user can easily control the relative emphasis to different scales and obtain the desired line drawing results. We demonstrate the improved PELs on a wide range of real-world objects.

Keywords Real-time line drawing · Object-space algorithm · Gradient operator · Non-photorealistic shading

This work has been supported by the National Research Foundation grant, which is administered by the Media Development Programme Office, MDA (IDMPO). Ying He is partially supported by AcRF 69/07. Long Zhang is partially supported by NSFC 60903085 and an Open Project Program of the State Key Lab of CAD&CG (A0905).

L. Zhang · Y. He (✉) · H.-S. Seah
Nanyang Technological University, Singapore, Singapore
e-mail: yhe@ntu.edu.sg

L. Zhang
e-mail: zhanglong@ntu.edu.sg

L. Zhang
e-mail: lzhang@cad.zju.edu.cn

H.-S. Seah
e-mail: ashsseah@ntu.edu.sg

L. Zhang
Hangzhou Dianzi University, Hangzhou, China

1 Introduction

Computer-generated line drawings are effective for conveying 3D shapes in a relatively succinct manner by ignoring less important details [19]. The past decade has witnessed a large amount of research efforts in this field, and many effective feature lines have been developed, e.g., suggestive contours [5], ridge-valley lines [16], apparent ridges [8], principal and suggestive highlights [4], photic extremum lines [23], demarcating curves [10], relief edges [11], and Laplacian lines [24], just to name a few. In the recent study, Cole et al. [3] investigated how artists' drawings of 3D shapes correlate with the mathematical properties of the shapes and known computer graphics line drawing techniques. They showed that most lines that do not overlap contours overlap large gradients of the image intensity. In particular, Canny edges [2], characterizing the significant changes of the image intensity, provide the strongest cues for artists to use to draw lines. This observation motivates a research direction that generalizes the feature detection from 2D images to 3D surfaces.

Xie et al. proposed photic extremum lines (PELs) on 3D surfaces [23]. Given a 3D surface S and the illumination function $I : S \to \mathbb{R}$, PELs are a set of points where the variation of illumination in the direction of its gradient reaches

the local maximum. Roughly speaking, PELs naturally generalize the Canny edge detector [2] to 3D surfaces. Compared to other view-dependent feature lines, PELs are more flexible by offering the users more degrees of freedom to achieve desired illustration effects. For example, the user can control the line drawings by specifying the light position, the number of light sources, and even the illumination models. In [23], the authors used a point light whose position coincides with the eye to convey the overall shape and a few directional lights to highlight or hide the details of the user-specified regions. Though very effective for conveying 3D shapes, PELs have two drawbacks that seriously limit their applications. First, PELs are computationally expensive due to the heavy involvement of the third- and fourth-order derivatives. Even with the most simple illumination model (diffuse reflection), PELs are still too slow to be applied to interactive applications. Second, for models with scattered or multiscale features, the users have to manually specify the regions to be lit by the directional lights. This must be done with great care since two directional lights are not allowed to overlap, otherwise the overlapped shading may lead to some undesired lines.

To overcome the aforementioned problems, this paper presents a novel computational framework that improves both the speed and quality of PELs. We develop a graphics processing unit (GPU)-based algorithm that can render PELs at least an order of magnitude faster than the method in [23] and enable PELs to be applied to interactive applications. Within the new framework, the users do not need to worry about the auxiliary lights since both overall shape and the details at different frequencies can be simultaneously rendered by a single non-photorealistic light. Thus, the users can completely ignore the tedious auxiliary lights and can generate the desired line drawings much more quickly and easily. Figure 1 shows an example that compares the new framework with the conventional PEL algorithm.

The specific contributions of this paper are as follows:

- We develop a GPU algorithm to compute PELs in real time. Our algorithm is based on a closed-form formula to compute the gradient on triangular meshes. Using this formula, the first-order derivative can be formulated as a simple weighted sum of a vertex's one-ring neighbors, where the weights can be precomputed and the high-order derivatives can also be computed recursively. The computation is highly parallel in nature and can be fully implemented using GPU. This leads to a GPU-based PEL extraction algorithm that is one order of magnitude faster than the method in [23].
- We introduce non-photorealistic shading into the PEL framework. Instead of extracting PELs from standard diffuse lighting, we propose to use a non-photorealistic, or enhanced, shading technique. With the enhanced shading, the lighting is computed at multiple scales that simultaneously convey overall shape and details at different frequencies. As a result, one can generate the desired line drawing results without any auxiliary lights.

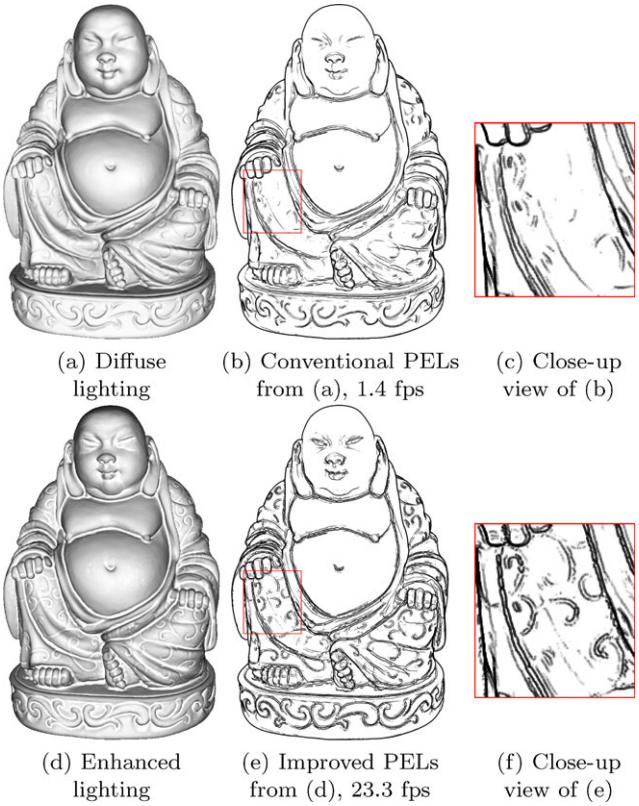


Fig. 1 Real-time computation of (PELs). The input Buddha model has fine details at the cloth, eyes, and base, etc. Using a single light and standard shading, the conventional PELs [23] can only depict the overall shape at 1.4 fps. Notice that many fine details are missing. In our proposed framework, we adopt an enhanced shading technique which effectively conveys the overall shape and details. As a result, the users can easily generate the desired line drawings using only one light. Furthermore, the proposed GPU-based algorithm can render PELs an order of magnitude faster than the method of Xie et al. [23]

The remainder of the paper is organized as follows. We review the related work of computer-generated line drawings and non-photorealistic shading in Sect. 2. Then we present the algorithm of real-time PEL computation in Sect. 3. In Sect. 4, we show how non-photorealistic shading can be employed to improve the PELs by bringing out the details at different frequencies. The experimental results are shown in Sect. 5. We conclude the paper in Sect. 6 and point out some future work.

2 Previous work

2.1 Computer-generated line drawings

There is an extensive literature on computer-generated line drawings and their applications. We refer the readers to the

SIGGRAPH course notes of Rusinkiewicz et al. [19] for a comprehensive list. The current techniques can be classified into two classes: object- and image-space. The former computes the features on the input 3D models directly and the latter computes them on the rendered image space.

The object-space feature lines are usually defined by a differential equation on the 3D surfaces. Silhouettes show the strongest cues with model-to-background distinction [6, 7]. However, silhouettes alone are quite limited in conveying shape, since they cannot capture the structure and complexity of the shape interior. DeCarlo et al. [5] proposed suggestive contours which naturally extend contours and convey the shape effectively. To address the problem that suggestive contours do not appear in the convex regions, DeCarlo and Rusinkiewicz [4] introduced highlight lines which complement contours and suggestive contours. Ridge-valley lines are powerful shape descriptors but independent of the view point [16]. Judd et al. [8] proposed apparent ridges which elegantly generalize ridge-valley lines with view-dependent features. Aiming at line drawing simplification, Ni et al. [15] proposed view-dependently controllable feature lines using preconstructed multiresolution mesh. Inspired by the Canny edge detector, Xie et al. [23] proposed PELs on 3D surfaces. Similar to the apparent ridge, the PEL is also a third-order feature line, and thus computationally expensive. Kolomenkin et al. [10] defined demarcating curves as the zeros of the normal curvature in the curvature gradient direction. In their recent work [11], Kolomenkin et al. considered the surface as an unknown smooth manifold, on top of which is placed a local height image. Then the relief edge is defined as the edges of the local height image. They showed that relief edges are continuous and smooth, and very promising for artifact illustration in archaeology [11]. Kalogerakis et al. [9] proposed a real-time and highly parallel method to compute curvatures and their derivatives for deforming objects. This method is ideal for deforming objects with temporal coherence by accurately predicting the curvatures and their derivatives. However, for static objects, this property does not hold, and the computation for surface curvatures and their derivatives is still expensive. Zhang et al. [24] proposed Laplacian lines that generalize the Laplacian of Gaussian (LoG) edge detector to 3D surfaces. They showed that Laplacian lines are computationally efficient because most expensive computations can be preprocessed, and the Laplacian line drawing algorithm is as simple as the conventional silhouette extraction algorithm.

Compared to the object-space algorithm, image-space line drawings are usually computationally efficient because they avoid the expensive computations of surface derivatives that most of the object-space algorithms require. Thus, image-space algorithms can be easily used to render animated models in real time. Saito and Takahashi [20] pioneered a method to extract feature lines using image processing techniques. Buchanan and Sousa [1] proposed the edge buffer data structure which enables highlighting various feature lines on a polygonal model. Raskar et al. [17] presented a novel NPR camera which detects depth edges using multiflash images. Winnemöller et al. [22] presented an automatic real-time video and image abstraction framework using difference-of-Gaussian edges. Lee et al. [13] proposed an algorithm to automatically extract lines at appropriate scales from abstract shading. Maciejewski et al. [14] used a well-studied image measurement to differentiate stippling images. Recently, Vergne et al. [21] proposed a novel image-space local shape descriptor to enhance surface depiction.

2.2 Enhanced surface depiction using lightings

There are a few research efforts that enhance surface depiction using lightings. Lee et al. [12] introduced geometry-dependent lighting that allows lighting parameters to be defined independently over an object or scene based on the local geometry. Inspired by the cartographic terrain relief, Rusinkiewicz et al. [18] presented exaggerated shading that dynamically adjusts the effective light position for different areas of the surface. Recently, Vergne et al. [21] presented an algorithm to enhance the shape depiction of 3D objects by locally warping the environment lighting around main surface features.

3 Computing PELs in real time

Given a 3D surface S and the illumination function $I : S \to \mathbb{R}$ defined on S, photic extremum lines (PELs) are a set of points on the 3D surface where the variation of illumination in the direction of its gradient reaches the local maximum [23], i.e.,

$$D_{\mathbf{d}} \|\nabla I\| = 0 \quad \text{and} \quad D_{\mathbf{d}} D_{\mathbf{d}} \|\nabla I\| < 0,$$

where ∇ denotes the surface gradient and $\mathbf{d} = \nabla I / \|\nabla I\|$.

3.1 Gradient operator

We observe that the performance of PELs heavily depends on the computation of the gradient at each vertex. Xie et al. [23] computed the gradient in a two-pass approach. First, the gradient vector for each face is computed. Second, the gradient at each vertex is computed by a weighted sum of the gradients of its adjacent faces.

In this paper, we propose an efficient single-pass approach to compute the per-vertex gradients. We are given a triangular mesh $M = (V, E, F)$, where V, E and F denote the vertex, edge, and face sets. Let $f : M \to \mathbb{R}$ be a

C^1 continuous scalar function defined on the mesh surface. Consider an arbitrary vertex $p \in V$. Let $\mathbf{u}$ and $\mathbf{v}$ be the principal axes that span the tangent plane of p, and $Nb(p)$ be the one-ring neighbors of p. The gradient $\nabla f(p)$ can be calculated by

$$\nabla f(p) = \sum_{q \in Nb(p) \cup \{p\}} \alpha(q) f(q) \mathbf{u} + \beta(q) f(q) \mathbf{v}, \qquad (1)$$

where $\alpha(q)$ and $\beta(q)$ are constant weights independent of the function f and can be precomputed. As a result, the gradient $\nabla f(p)$ is just a weighted sum of $f(p)$ and its one-ring neighbors. Furthermore, the high-order derivatives can be computed recursively using (1). A detailed derivation of (1) can be found in the Appendix.

3.2 Computing PELs in GPU

Computing vertex gradients using (1) is particularly suitable for implementation in GPUs. First, the computation is completely parallel, i.e., the gradient vector of each vertex is calculated individually. In addition, there is no data scattering throughout the computation.

We implement the gradient computation using a standard graphics pipeline, with an approach similar to GPU particle simulation. For each vertex, we draw a single point. Indices of the neighboring vertices Nb, and the coefficients α, β are transferred to the pipeline as constant vertex attributes. The dynamic scalar function f is stored in a texture and sampled by the fragment program. Each fragment computes the gradient vector of the corresponding vertex using (1) and outputs the result to the frame buffer. In this way, the gradient vector of each vertex is stored in one pixel of the frame buffer and is read by texture sampling in the next pass.

The complete pipeline of PEL rendering consists of four passes as shown in Fig. 2.

1. Compute the illumination I by a dot product of the vertex normal $\mathbf{n}$ and the viewing vector $\mathbf{v}$.
2. Compute ∇I using (1) as explained above.
3. Compute $D_{\mathbf{d}} \|\nabla I\| = \mathbf{d} \cdot \nabla(\|\nabla I\|)$ in a similar fashion, i.e., taking $\|\nabla I\|$ as the scalar field.
4. Compute and render the iso-lines satisfying $D_{\mathbf{d}} \|\nabla I\| = 0$.

The iso-line computation in the fourth step is done on a per-face basis, with a simple geometry shader. More concretely, we render a triangle for each facet and use the geometry shader to compute the location of the iso-line segment in the triangle and output the line segment to the graphics pipeline. The inequality constraints are also performed in the geometry shader. Similar to the software implementation, the test is performed only in triangles intersecting the iso-lines. The normalized $\|\nabla I\|$ is output to the alpha component of the image color, and thresholding is done via alpha test.

The implementation is memory efficient in that all constant data reside in high-speed static memory, and only the dynamic scalar field is stored in texture memory and read by texture sampling. In addition, the memory consumption is reasonable. For each vertex, we only need to store the neighbor indices and corresponding blending weights besides vertex positions and normals. On modern GPUs, we can process large-scale models with millions of vertices.

4 PELs based on enhanced shading

The major goal of computer-generated line drawings is to convey both the overall shape and fine details as clearly as possible. In the original PELs paper [23], Xie et al. assumed that the surfaces have only diffuse materials and the lightings include a (required) main light and (optional) auxiliary

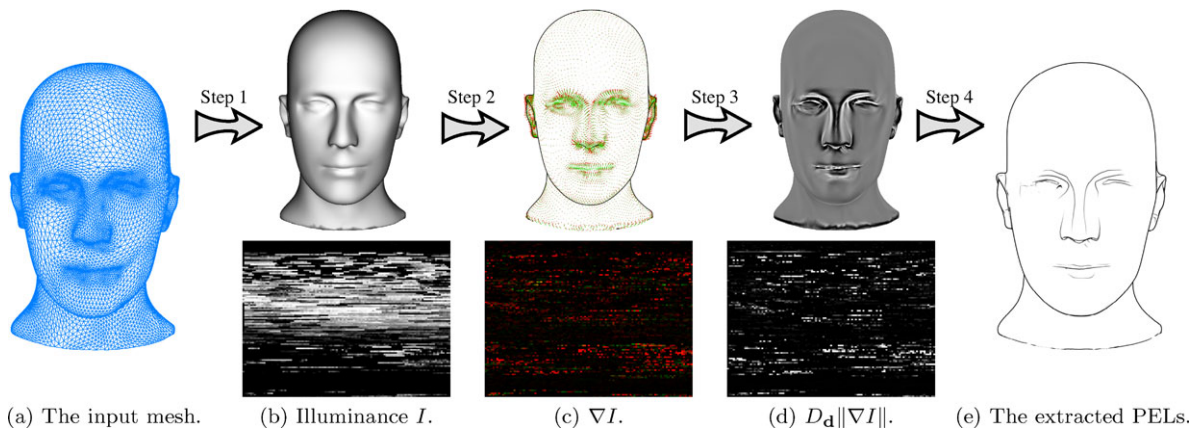


Fig. 2 Computing PELs in GPU consists of four steps, i.e., computing I, ∇I, $D_{\mathbf{d}}\|\nabla I\|$, and extracting lines. For each of the first three steps, the temporary computation results are stored in a 2D texture, where each vertex is stored in one pixel. The last step extracts the iso-line of $D_{\mathbf{d}}\|\nabla I\|$ with a simple geometry shader

lights. The main light is used to convey the overall shape and the auxiliary lights to highlight or hide the fine details of the user-specified regions. As mentioned earlier, for complex shapes with multiscaled features, it is usually tedious to control these auxiliary lights since two auxiliary lights (directional lights) are not allowed to overlap, otherwise, some undesired lines will show in the overlapping region.

In this paper, we show that the constraint of lambertian illumination in PEL rendering is unnecessary. On the contrary, we can take advantage of the recently developed enhanced shading techniques to achieve better line drawing results. Compared with standard illumination models, the enhanced shadings better convey the model details. Taking an enhanced shading image as input, the extracted PELs should better depict the fine details accordingly.

Figure 3 shows an example comparing the PELs based on standard diffuse shading and enhanced shading. The enhanced shading highlights the rich details of the model, such as on the legs, eyes, and armor. Clearly, the enhanced shading leads to better rendering results than the standard shading.

Incorporating PELs with an enhanced shading technique is straightforward: simply replace the diffuse shading computation (in the first pass of the PEL rendering pipeline) with the enhanced shading, and the rest of the steps remain the same. Theoretically, any enhanced shading method can be used for rendering PELs. In practice, we adopted a simple scheme that exaggerates the diffuse shading by scaling a factor related to the surface curvature, as proposed by [21]. Let I_i be the diffuse illumination at vertex i, and k_u and k_v be the principal curvatures, then the enhanced illumination I_i' is computed as $I_i' = I_i \cdot (\lambda_u \lambda_v)^\gamma$ and $\lambda_{u|v} = \tan(\arctan(\alpha k_{u|v})/6 + \pi/4)$, where α and γ are parameters controlling the degree of exaggeration. By adjusting these parameters, different PEL rendering results can be achieved with the same model. In this way, our framework offers an intuitive interaction scheme to generate desired line drawing results.

The GPU implementation of enhanced shading is rather simple. We need only modify the vertex program for the first pass to perform the illumination scale with the above equation and output the scaled illumination to the frame buffer.

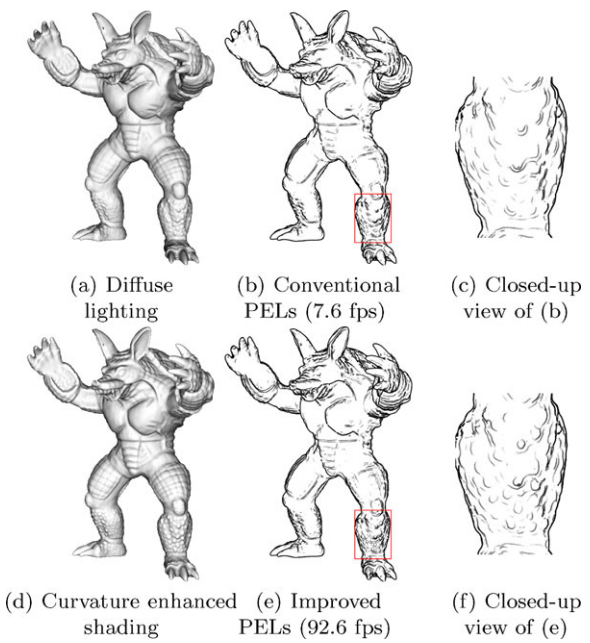


Fig. 3 With standard rendering methods it is difficult to depict every detail on the 3D surfaces. As a result, the generated PELs can only convey the overall shape, missing the fine details. The enhanced shading, on the contrary, is effective for depicting the overall shape and also bringing out the details simultaneously. Clearly, enhanced shading leads to a more clear line drawing than standard diffuse shading

5 Experimental results

We implemented the GPU algorithm using the Cg shading language and tested our prototype system for a wide range of 3D models on a PC with 3.0 GHz CPU and NVIDIA 285 GTX GPU. The statistics are shown in Table 1.

For all the tested models, the GPU implementation is at least 11 times faster than the CPU implementation. For the Buddha model, the performance speed-up is as high as 16.5 times. The speed-up is more obvious for large models because the GPU algorithm involves multiple passes, and the state changes between different passes incur constant overhead time.

Figure 4 demonstrates that the improved PEL framework is flexible by giving the users more degrees of freedom to enhance the shading globally and locally.

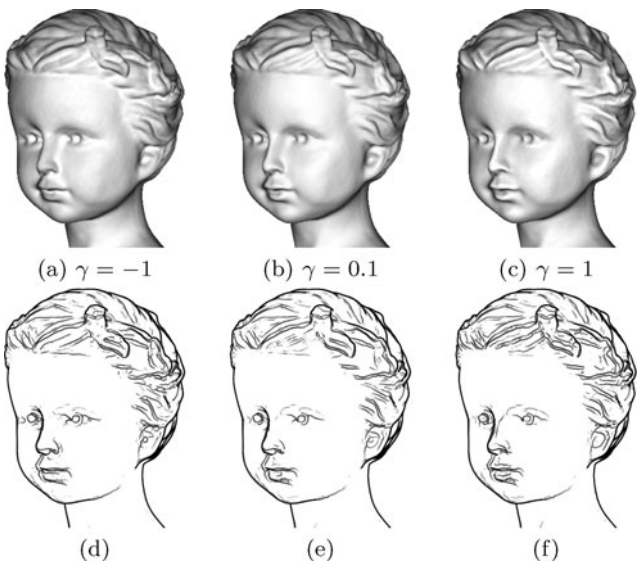


Fig. 4 By tuning the parameters in enhanced shading, different PEL rendering effects can be achieved. Note the different lines on the nose and around the eyes. Parameter α is set to 1 for all images

Table 1 Performance comparison of various line drawing algorithms. #△: number of triangles. The third through seventh columns show the rendering speed in fps: Suggestive contours (SC); Apparent ridges (AR); Laplacian lines (LL); Conventional PEL (PEL); GPU-accelerated PEL (GPEL)

Models	#△	SC	AR	LL	PEL	GPEL
Armadillo	$330K$	33.3	8.9	37.0	7.6	92.6
Bimba	$1M$	11.2	2.9	12.5	2.5	32.2
Buddha	$1.5M$	6.7	1.7	7.4	1.4	23.3
Bunny	$144K$	76.8	20.6	85.4	17.6	213.4
Kid	$511K$	23.8	6.9	26.5	5.9	66.6
Lion	$400K$	28.6	7.0	31.7	6.0	69.0
Mannequin	$23K$	455	143	505	122	1450
Sculpture	$247K$	45.5	9.5	50.5	8.1	119

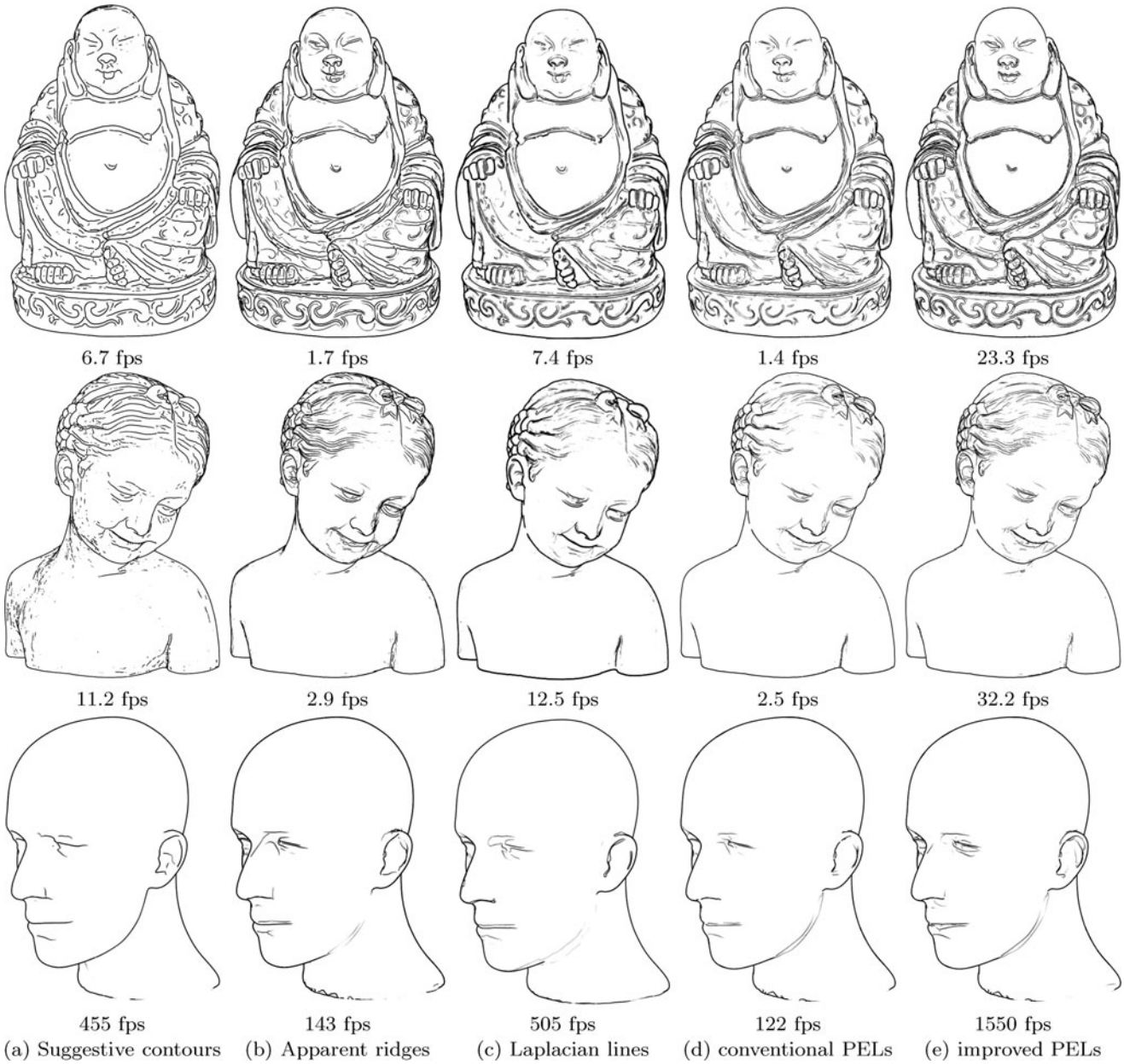


Fig. 5 Comparison of the improved PELs and other feature lines. Among all feature lines, the improved PELs best depict the model details, e.g., the model eyes of the Buddha model and the Mannequin model

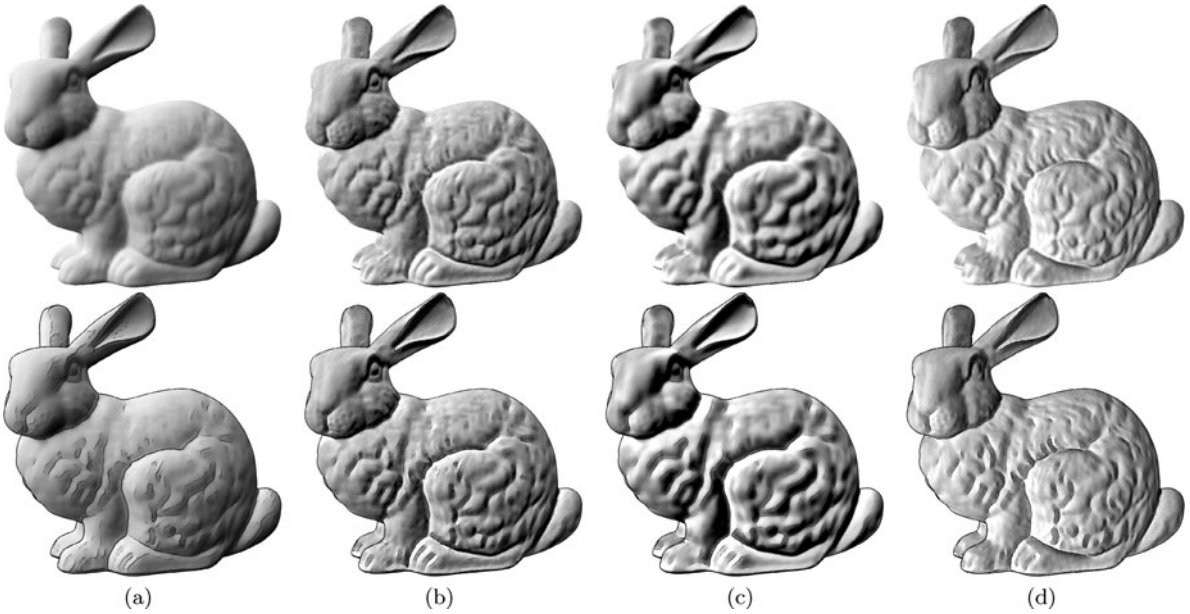


Fig. 6 Comparison of conventional PELs and PELs extracted from exaggerated shading. (**a**) Conventional PELs based on diffuse lighting; (**b**) improved PELs based on exaggerated shading; (**c**) exaggerated shading with a different parameter, where coarse-level shading is made more important; (**d**) exaggerated shading with a different light direction

Figure 5 compares the proposed approach with several line drawing algorithms including the original PEL algorithm. Using the enhanced shading image as input, PEL can depict the details better than original PEL, which uses diffuse lighting. For example, the new PEL algorithm more effectively depicts the hairs of the Bimba model and the eyes of the Mannequin model.

As mentioned before, theoretically, any shading technique can be used to generate PELs. Besides the curvature enhanced shading, we also tried several other shading algorithms. Figure 6 shows an example using exaggerated shading [18]. The exaggerated shading is composited by illumination calculated at various scales. The user can control the weights of different scales by adjusting a few intuitively, which results in different shading effects. Accordingly, different PEL rendering results can be obtained.

Figure 7 compares conventional PELs with PELs based on mean-curvature shading. Standard diffuse lighting cannot highlight every detail on the model surface. As a result, conventional PELs based on the diffuse lighting miss some important features of the model. In contrast, the mean-curvature shading clearly convey most of the features, from which a better PEL rendering result is generated.

Another example is shown in Fig. 8. The handle model is scanned from an ancient object and is very noisy. Standard diffuse lighting does not convey the features clearly. Accordingly, the resulting PELs are totally unsatisfactory. The mean-curvature shading effectively suppresses the noises and conveys the major features of the model, and the resulting PELs clearly depict the characters on the model surface.

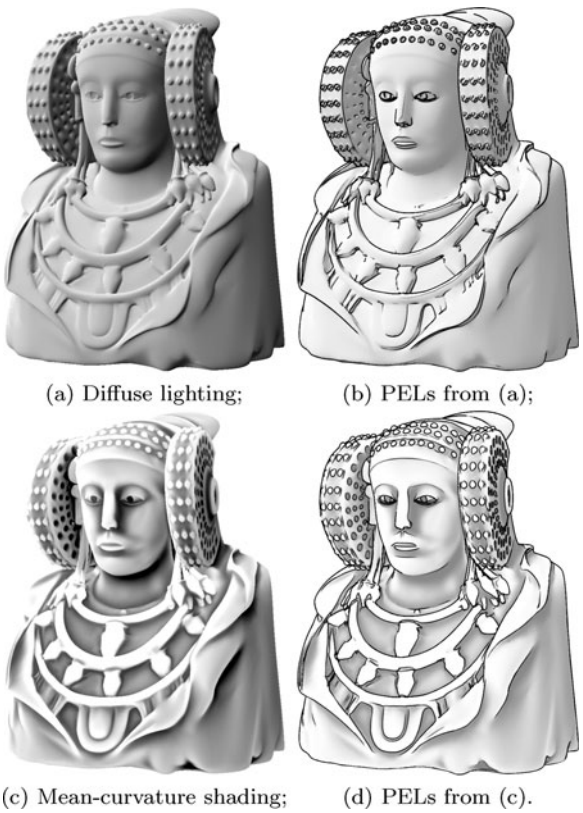


Fig. 7 Comparison of conventional PEL and PELs extracted from mean-curvature shading. The mean-curvature shading (**c**) highlights the fine details better than standard diffuse lighting (**a**). Consequently, PELs based on mean-curvature shading (**d**) depict various features of the model better than the conventional PELs (**b**), e.g., the eyebrow, nose, jewelry, etc.

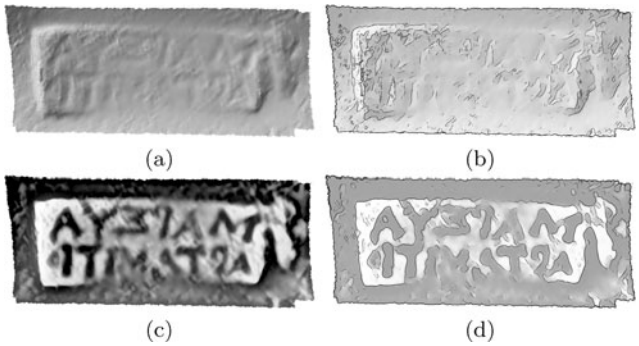


Fig. 8 Another comparison of conventional PELs and PELs based on mean-curvature shading. The input model is noisy, see (**a**). The conventional PELs do not provide any interesting information, see (**b**). The mean-curvature shading provides better contrast than the diffuse shading, see (**c**). As a result, the enhanced PELs effectively depict the characters on the model surface, see (**d**)

One distinct advantage of PELs, compared to other feature lines, is that PELs can be naturally combined with any enhanced shading technique to convey shapes simultaneously. The reason is that PELs are generalized from the Canny edge detector. Therefore, given an arbitrary shading effect, PELs extracted from it mostly coincide with the boundary of the shading image. This effect can be seen in Fig. 6–8.

6 Conclusions and future work

This paper presented a novel computational framework to improve both the performance and quality of photic extremum lines (PELs). We derived a simple and closed-form formula for the gradient operator such that various orders of derivatives can be computed efficiently and in parallel using GPU. We developed a GPU-based PEL extraction algorithm that is an order of magnitude faster than the one in [23]. Based on the observation that standard graphics rendering cannot depict the overall shape and the details at different frequencies simultaneously, we adopted non-photorealistic shading into the PELs framework and showed that the user can easily control the relative emphasis to different scales and get the desired line drawings easily. Finally, by testing a wide range of real-world models, we demonstrated that the improved PELs are promising for depicting the surface details in real time.

In this paper, we implement the GPU-based PEL extraction algorithm using a standard graphics pipeline. In the future, we would like to try to implement it using the more advanced CUDA framework. We will also develop GPU-based extraction algorithms for other classical feature lines. In addition, we will try to improve the PEL rendering quality with more shading techniques.

Acknowledgement The models are courtesy of Stanford University, Aim@Shape, and artist3d.com.

Appendix

In this section, we derive the closed-form formula to compute the gradient on the triangular mesh.

Let $M = (V, E, F)$ be the triangular mesh, where V, E, F are the vertex, edge, and face sets, respectively. A smooth scalar function $f : M \to \mathbb{R}$ is defined on the vertex set.

Given an arbitrary point $v_i \in V$, let $F_k = (k_1, k_2, k_3)$, $k = 1, \ldots, m$, denote the neighboring triangles that are incident to v_i, where k_i, $i = 1, 2, 3$, denotes the vertex index. The principal axes of the local coordinate system of triangle F_k are $\mathbf{x}_k$ and $\mathbf{y}_k$, respectively. Then the per-face gradient of the triangle F_k, $\nabla f|_{F_k}$, is given by

$$\nabla f|_{F_k} = \frac{1}{A_k}(\mathbf{x}_k, \mathbf{y}_k) \begin{pmatrix} y_2^k - y_3^k & y_3^k - y_1^k & y_1^k - y_2^k \\ x_2^k - x_3^k & x_3^k - x_1^k & x_1^k - x_2^k \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix},$$

where $A_k = (x_1^k y_2^k - y_1^k x_2^k) + (x_2^k y_3^k - y_2^k x_3^k) + (x_3^k y_1^k - y_3^k x_1^k)$ is twice the signed area of triangle F_k.

Let $\mathbf{u}_i$ and $\mathbf{v}_i$ be the principal axes at vertex v_i, and the projection of $(\mathbf{u}_i, \mathbf{v}_i)$ on face F_k be $(\mathbf{u}_i^k, \mathbf{v}_i^k)$, then the vertex gradient $\nabla f(v_i)$ is computed by

$$\nabla f(v_i) = (\mathbf{u}_i, \mathbf{v}_i) \sum_{k=1}^{m} w_i^k \left(\left(\mathbf{u}_i^k\right)^T, \left(\mathbf{v}_i^k\right)^T \right)^T \cdot \nabla f|_{F_k}.$$

The blending weight w_i^k is the same as defined in [23]. Define

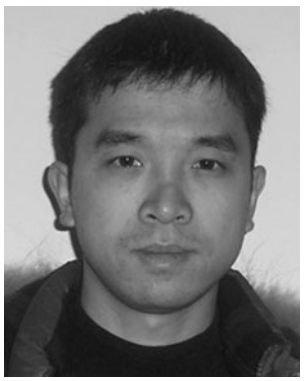
$$\begin{aligned} \mathbf{A}_k &\triangleq \begin{pmatrix} a_{k_1} & a_{k_2} & a_{k_3} \\ b_{k_1} & b_{k_2} & b_{k_3} \end{pmatrix} \\ &= \frac{w_i^k}{A_k} \begin{pmatrix} \mathbf{u}_i^k \cdot \mathbf{x}_k & \mathbf{u}_i^k \cdot \mathbf{y}_k \\ \mathbf{v}_i^k \cdot \mathbf{x}_k & \mathbf{v}_i^k \cdot \mathbf{y}_k \end{pmatrix} \\ &\quad \times \begin{pmatrix} y_2^k - y_3^k & y_3^k - y_1^k & y_1^k - y_2^k \\ x_2^k - x_3^k & x_3^k - x_1^k & x_1^k - x_2^k \end{pmatrix}, \end{aligned}$$

then

$$\begin{aligned} \nabla f(v_i) &= (\mathbf{u}_i, \mathbf{v}_i) \cdot \sum_{k=1}^{m} \mathbf{A}_k \cdot (f_{k_1}, f_{k_2}, f_{k_3})^T \\ &= \left(\sum_{k=1}^{m} \sum_{j=1}^{3} a_{k_j} f_{k_j} \right) \mathbf{u}_i + \left(\sum_{k=1}^{m} \sum_{j=1}^{3} b_{k_j} f_{k_j} \right) \mathbf{v}_i \\ &= \sum_{q \in Nb(v_i) \cup \{p\}} \alpha(q) f(q) \mathbf{u}_i + \beta(q) f(q) \mathbf{v}_i. \end{aligned}$$

References

1. Buchanan, J.W., Sousa, M.C.: The edge buffer: a data structure for easy silhouette rendering. In: NPAR '00, pp. 39–42. ACM, New York (2000)
2. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**, 679–698 (1986)
3. Cole, F., Golovinskiy, A., Limpaecher, A., Barros, H.S., Finkelstein, A., Funkhouser, T., Rusinkiewicz, S.: Where do people draw lines? ACM Trans. Graph. **27**(3) (2008)
4. DeCarlo, D., Rusinkiewicz, S.: Highlight lines for conveying shape. In: Proceedings of NPAR, pp. 63–70 (2007)
5. DeCarlo, D., Finkelstein, A., Rusinkiewicz, S., Santella, A.: Suggestive contours for conveying shape. In: Proceedings of ACM SIGGRAPH, pp. 848–855 (2003)
6. Gooch, B., Sloan, P.J., Gooch, A., Shirley, P., Riesenfeld, R.F.: Interactive technical illustration. In: Proceedings of ACM Symposium on Interactive 3D Graphics, pp. 31–38 (1999)
7. Hertzmann, A., Zorin, D.: Illustrating smooth surfaces. In: Proceedings of ACM SIGGRAPH, pp. 517–526 (2000)
8. Judd, T., Durand, F., Adelson, E.: Apparent ridges for line drawing. In: Proceedings of ACM SIGGRAPH (2007)
9. Kalogerakis, E., Nowrouzezahrai, D., Simari, P., McCrae, J., Hertzmann, A., Singh, K.: Data-driven curvature for real-time line drawing of dynamic scene. ACM Trans. Graph. **28**(1) (2009)
10. Kolomenkin, M., Shimshoni, I., Tal, A.: Demarcating curves for shape illustration. ACM Trans. Graph. **27**(5) (2008)
11. Kolomenkin, M., Shimshoni, I., Tal, A.: On edge detection on surfaces. In: CVPR'09, pp. 2767–2774 (2009)
12. Lee, C.H., Hao, X., Varshney, A.: Geometry-dependent lighting. IEEE Trans. Vis. Comput. Graph. **12**(2), 197–207 (2006)
13. Lee, Y., Markosian, L., Lee, S., Hughes, J.F.: Line drawings via abstracted shading. ACM Trans. Graph. **26**(3), 18 (2007)
14. Maciejewski, R., Isenberg, T., Andrews, W.M., Ebert, D.S., Sousa, M.C., Chen, W.: Measuring stipple aesthetics in hand-drawn and computer-generated images. IEEE Comput. Graph. Appl. **28**(2), 62–74 (2008)
15. Ni, A., Jeong, K., Lee, S., Markosian, L.: Multi-scale line drawings from 3D meshes. In: Proceedings of ACM Symposium on I3D, pp. 133–137 (2006)
16. Ohtake, Y., Belyaev, A., Seidel, H.: Ridge-valley lines on meshes via implicit surface fitting. In: Proceedings of ACM SIGGRAPH, pp. 609–612 (2004)
17. Raskar, R., Tan, K.H., Feris, R., Yu, J., Turk, M.: Non-photorealistic camera: depth edge detection and stylized rendering using multi-flash imaging. In: SIGGRAPH Course Notes (2005)
18. Rusinkiewicz, S., Burns, M., DeCarlo, D.: Exaggerated shading for depicting shape and detail. ACM Trans. Graph. **25**(3), 1199–1205 (2006)
19. Rusinkiewicz, S., Cole, F., DeCarlo, D., Finkelstein, A.: Line drawings from 3D models. In: SIGGRAPH Course Notes (2008)
20. Saito, T., Takahashi, T.: Comprehensible rendering of 3D shapes. Proc. ACM SIGGRAPH **24**(4), 197–206 (1990)
21. Vergne, R., Pacanowski, R., Barla, P., Granier, X., Schlick, C.: Light warping for enhanced surface depiction. ACM Trans. Graph. **28**(3) (2009)
22. Winnemöller, H., Olsen, S.C., Gooch, B.: Real-time video abstraction. ACM Trans. Graph. **25**(3), 1221–1226 (2006)
23. Xie, X., He, Y., Tian, F., Seah, H.S., Gu, X., Qin, H.: An effective illustrative visualization framework based on photic extremum lines (PELs). IEEE Trans. Vis. Comput. Graph. **13**(6), 1328–1335 (2007)
24. Zhang, L., He, Y., Xie, X., Chen, W.: Laplacian lines for real-time shape illustration. In: Proceedings of ACM Symposium on Interactive 3D Graphics and Games (I3D '09), pp. 129–136 (2009)



Long Zhang is an assistant professor in the School of Computer Science and Technology, Hangzhou Dianzi University. He received his Ph.D. degree in 2008 from the Department of Mathematics, Zhejiang University. His current research interests include nearly perfect reconstruction (NPR), expressive rendering, and visualization.

Ying He is an assistant professor at the School of Computer Engineering, Nanyang Technological University. He received his Ph.D. degree in Computer Science from Stony Brook University in 2006. His research interests include computer graphics, computer-aided design, and scientific visualization. He is particularly interested in the problems which require geometric analysis and computation. For details, visit http://www.ntu.edu.sg/home/yhe.

Hock-Soon Seah is a professor and director of the gameLAB at the School of Computer Engineering at Nanyang Technological University (NTU), Singapore. Concurrently, he is also a co-director of the NTU Institute for Media Innovation.