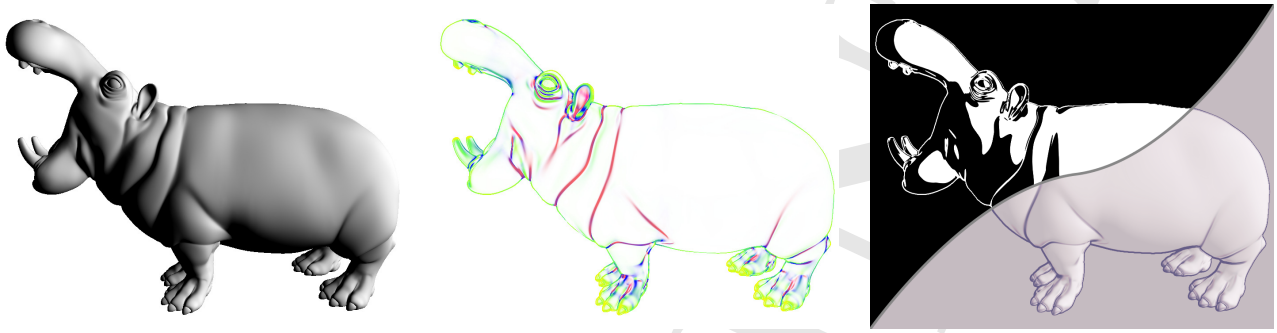
Apparent relief: a shape descriptor for stylized shading

Romain Vergne, Pascal Barla, Xavier Granier, Christophe Schlick

Apparent relief: a shape descriptor for stylized shading

Romain Vergne Pascal Barla Xavier Granier Christophe Schlick

INRIA Bordeaux University



(a) (b) (c)

Figure 1: ***Depicting shape through shading:*** *Starting from a 3D model (a), we extract our view-dependent shape descriptor (b) - (see Figure 2 for color code). We then manipulate shading based on* continuous *shape cues selected using the descriptor, and create various styles such as minimal shading and cartoon shading (c).*

Abstract

Shape depiction in non-photorealistic rendering of 3D objects has mainly been concerned with the extraction of contour lines, which are generally detected by tracking the *discontinuities* of a given set of shape features varying on the surface and/or the picture plane. In this paper, we investigate another approach: the depiction of shape through shading. This technique is often used in scientific illustration, comics, cartoon animation and various other artwork. A common method consists in indirectly adapting light positions to reveal shape features; but it quickly becomes impractical when the complexity of the object augments. In contrast, our approach is to directly extract a set of shape cues that are easily manipulated by a user and re-introduced during shading. The main problem raised by such an approach is that shape cues must be identified in a *continuous* way in image space, as opposed to line-based techniques. Our solution is a novel view-dependent shape descriptor called Apparent Relief, which carries pertinent continuous shape cues for every pixel of an image. It consists of a combination of object- and image-space attributes. Such an approach provides appealing properties: it is simple to manipulate by a user, may be applied to a vast range of styles, and naturally brings levels-of-detail functionalities. It is also simple to implement, and works in real-time on modern graphics hardware.

1 Introduction

Shape depiction is an important dimension of image creation for a variety of reasons. In scientific illustration, shape depiction techniques are used to remove possible ambiguities in visual interpretation. They are often seen as processes that highlight the most salient characteristics of an object's shape. This is often done at the expense of other details which are removed because they are irrelevant to the information the image is supposed to convey [Wood 1994]. In paintings and drawings, shape depiction techniques may be used in more subtle ways, and for other purposes. For example, they might help understand relationships between characters and background; clarify a representation that makes use of drastically simple shading rules; or even portray hidden portions of surfaces like in cubist paintings.

Providing an intuitive and efficient control over the depiction of 3D objects' shape in Computer Graphics is thus of primary importance. Many previous approaches focus on a single body of techniques: line-based rendering. The goal of such techniques is to generate some *shape cues* to depict the essential characteristics of an object that correspond to *discontinuities* of shape features. For instance, contour lines may represent discontinuities of depth, curvature, orientation, object IDs, etc... Line-based techniques have been used in a variety of applications, from architecture to video games. They represent only a subset of shape depiction techniques though. In particular, many artists rather depict an object's shape through shading: e.g. they may use gradients, which are other kinds of shape cues that correspond to *continuous* variations of shape features. Some reasons for the choice of such a style are that it allows the artist to convey more subtle information about shape, and that it is integrated seamlessly into conventional lighting [Hogarth 1991].

Previous methods often imitated traditional illustrations to depict shape through shading, e.g. by creating artificial light sources that indirectly convey shape cues; unfortunately, such methods quickly become cumbersome to manipulate in practice. For this reason, we rather take the approach of directly extracting and manipulating shape cues. The main issue is that these cues can no longer be defined as sharp *discontinuities* as in line-based rendering, because when integrated into shading, they would have little influence. Instead, we seek a set of *continuous* shape cues that have to be defined for each pixel of an image. Expressing them directly from the 3D data that defines object's shape would be rather complex. Therefore, an intermediate representation, that we call a shape descriptor, is needed to assist the user.

The main contribution of this paper is to introduce a view-dependent shape descriptor called Apparent Relief, from which continuous shape cues are easily extracted, and which gives rise to stylized shading-based shape depictions. By construction, it is

free of temporal coherence artifacts and naturally leads to *automatic* Levels-of-Detail effects: when the object gets closer or farther from the point of view, finer or coarser shape cues are revealed respectively. Our approach runs in real-time on modern graphics hardware, and is also simple to implement. We illustrate its potential using four shading styles: cel shading, cartoon shading, minimal shading, and exaggerated shading.

2 Previous Work

Many previous approaches focus on line-based renderings to depict shape in a non-photorealistic way. A conventional method consists in analyzing differential geometry to identify maxima and minima of curvature (i.e., third-order discontinuities), such as ridges and valleys [Ohtake et al. 2004]. Other techniques take the point of view into account to extract either silhouettes [Hertzmann 1999] or suggestive contours [DeCarlo et al. 2003; DeCarlo et al. 2004; DeCarlo and Rusinkiewicz 2007], which extend and anticipate silhouettes. A line-based rendering can also be obtained from pure image-space attributes, e.g. by tracking discontinuities of depth, object IDs [Saito and Takahashi 1990; Hertzmann 1999], or directions of normal vectors [Decaudin 1996; Nienhaus and Döllner 2005].

With object-space techniques, contour lines are explicitly extracted as vector primitives, hence opening their rendering to stylization. This is not a trivial process though, as the extraction of a *temporally coherent* stylization is a complex task [Kalnins et al. 2003]. Moreover, the number and location of lines need to be adapted based on the viewing distance, a process similar to the creation of Levels of Detail (LODs). Some solutions work by using geometry simplification as a pre-process [Jeong et al. 2005; Ni et al. 2005], but there is currently no method to deal with this issue dynamically. Image-based methods avoid the need for geometry simplification, and are thus naturally adapted to LODs: as the number and location of extracted lines directly depend on the projected size of the 3D object, LODs are managed automatically. On the other hand, since lines are only identified as pixels, they do not have direct access to object-space attributes such as surface convexity, and are not easily stylized. Our approach combines advantages of object- and image-space techniques.

Recently, a new method called Apparent Ridges [Judd et al. 2007] has been introduced to provide a generalization to previous line-based rendering techniques. It relies on the idea that even object-space contour lines should be extracted by tracking the discontinuities of a view-dependent shape descriptor. In the case of line-based shape depiction, apparent ridges provide quite convincing results. However, they cannot be directly employed to fulfill our goal of shading-based shape depiction: they do not provide continuous shape cues, are ill-defined at silhouettes and require further specific LOD mechanisms depending on the viewing distance. This work inspired ours though, and we give comparisons with Apparent Ridges in Section 3.1.

Another approach to shape depiction is to rely on a notion of *accessibility*. The original accessibility shading method [Miller 1994] relies on a geometric shape descriptor: accessibility is defined as the maximum radius of a sphere touching the point of interest without intersecting other portions of the surface. Another descriptor has gained a lot of interest with the so-called ambient occlusion technique [Zhukov et al. 1998]: in this case, the amount of irradiance reaching a point characterizes its accessibility. Unfortunately, these methods are not view-dependent (e.g., ignoring silhouettes), and the shape cues they convey only consist of a scalar value at each point which brings little information in deep concavities. Moreover, both accessibility methods usually demand large precomputation times.

Many NPR shading techniques have also been presented in previous work (e.g., [Gooch et al. 1999; Barla et al. 2006]). However, none of them proposed to depict object shape explicitly (i.e., the user has no direct control over shape cues). Moreover, they are often restricted to specific styles. Other approaches rather modify lighting [Lee and Hao 2006; Rusinkiewicz et al. 2006], as is the case of exaggerated shading: the idea is to locally adjust the light direction over different areas of a surface, revealing details usually only seen in places where light reaches a grazing angle. The method does not explicitly extract a shape descriptor though, so that the control on the user side is restricted to the exaggeration of any detail on the surface, and stylization is limited to representing tone changes. In contrast, our approach allows to directly manipulate shape cues through a view-dependent shape descriptor, which in turn allows us to control exaggerated shading more intuitively.

3 Apparent relief descriptor

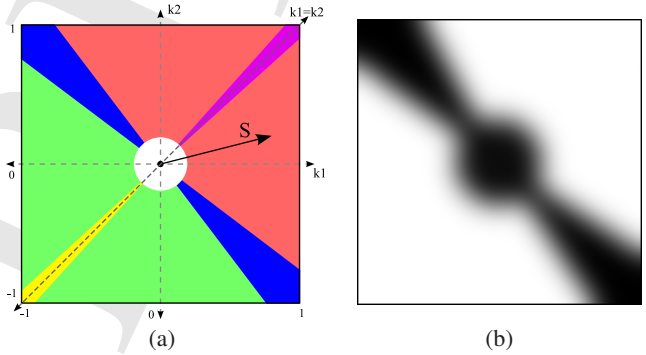


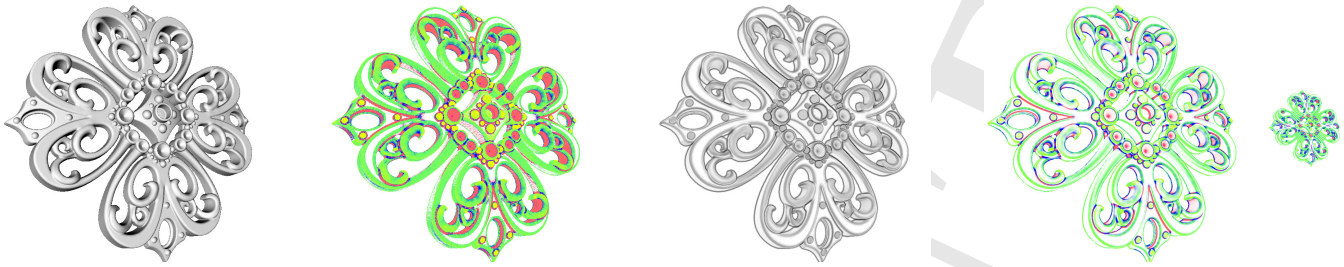
Figure 2: ***Shape descriptor domain****: In (a) we show the color code used throughout the figures of the paper to visualize our shape descriptor: planes are in white, caps (in yellow) and cups (in pink) are located around the symmetry axis, and saddles (in blue) separate convex (in green) from concave regions (in red). The axis of the parameter space correspond to principal curvatures k_1 and k_2. In (b), we show an example of Apparent Relief Map (ARM). Using this texture, shape cues are easily selected: here, surface points corresponding to planar- and saddle-like regions are filtered out, as explained in Section 4.*

Our solution to shading-based shape depiction is to manipulate continuous cues through a shape descriptor inspired from [Koenderink and van Doorn 1992]: for each pixel, we extract a descriptor $\vec{\mathbf{S}}$, which corresponds to a vector in the shape descriptor domain of Figure 2(a). The direction of $\vec{\mathbf{S}}$ gives information about surface convexity, while its length gives information about surface curvedness. The user is then able to select specific shape cues via a dedicated texture as the one shown in Figure 2(b).

However, our Apparent Relief Descriptor differs from the one defined in [Koenderink and van Doorn 1992] as it makes use of two sets of shape attributes: **First**, convexity information is directly computed in object space using differential geometry; **Second**, curvedness information is computed in image-space, incorporating view-dependency and LOD functionalities. The entire process is depicted in Figure 3.

3.1 Object-space information

The shape descriptor of Koenderink and van Doorn [1992] describes an arbitrary 3D surface at any point. The intuition behind it is given in Figure 2(a): the axes correspond to principal curvatures k_1 and k_2, so that any surface point can be assigned a shape vector $\vec{\mathbf{S}}$ in such a shape descriptor domain. Note that k_1 and k_2 are considered to be in the unit range $[-1,1]$, which requires some normaliza-



(a) (b) (c) (d)

Figure 3: ***Extracting the Apparent Relief Descriptor****: Given an input 3D object (a), we first analyze object-space shape attributes to extract convexity information (b). Then we extract curvedness information from normal variations in image-space (c) and combine both sources in a single shape descriptor: the Apparent Relief (d) - see Figure 2(a) for color code. Note the automatic LODs obtained thanks to image-space measurements in the rightmost image.*

tion step, detailed below. The shape vector is conveniently defined by its direction $\vec{\mathbf{D}}$ which corresponds to convexity information, and its length $\mathbf{L}$, which corresponds to curvedness information. Given $\vec{K} = (k_1, k_2)$, they are given by $\vec{\mathbf{D}} = \vec{K}/||\vec{K}||$ and $\mathbf{L} = ||\vec{K}||/2$. Note the symmetry around the first diagonal in Figure 2: this is due to the arbitrary assignment of k_1 and k_2 (the axis of symmetry corresponding to $k_1 = k_2$). Note also that $\vec{\mathbf{D}}$ is undefined for $||\vec{K}|| = 0$. It corresponds to the center of the shape descriptor domain, and only occurs in locally planar regions on the surface (we will come back to this singularity in Section 3.3).

The most straightforward approach is to compute the shape descriptor at each point of the surface. However, we must first remap each of the principal curvatures k_1 and k_2 to the unit range $[-1,1]$. To this end, we use the scaling function $\mathscr{S}_\alpha(x)$ described in the Appendix. There is an important limitation to this straightforward use of $\vec{\mathbf{S}}$ though: it does not consider view-dependency at all, for instance leaving out most of the information around silhouettes. Note that the view-dependent curvature operator introduced in [Judd et al. 2007] cannot be used to deal with this issue, as the corresponding principal curvature directions are not orthogonal, a requirement for the use of Koenderink's shape descriptor.

Our solution to this problem of view-dependency is to define a hybrid shape descriptor combining object-space and image-space attributes. More precisely, the main idea is to directly compute the shape vector's direction $\vec{\mathbf{D}}$ in object-space, while deferring the computation of its length $\mathbf{L}$ to image-space. This approach is somehow similar to the method used in [Judd et al. 2007], as they also use pure object-space attributes to discriminate between ridges and valleys in the very end of their pipeline. Our approach is quite different though, as we compute a richer convexity information (not only a binary "ridge or valley" flag), and for every visible surface point (not only onto discontinuities).

In practice, we compute a curvature tensor for each vertex of a triangle mesh using the algorithm in [Rusinkiewicz 2004]. To ensure that curvatures are continuously interpolated per pixel on the GPU, we sort curvatures according to $k_1 \geq k_2$. Thanks to the symmetry axis, only the lower-right triangular area of the shape descriptor domain has to be considered. However, note that in general, k_1 and k_2 do *not* respectively correspond to maximum and minimum curvatures.

At this stage, $\vec{\mathbf{D}}$ is computed independently for every pixel in the picture, so that sharp transitions of $\vec{\mathbf{D}}$ values may happen in image-space, e.g. between convex and concave regions. It occurs most notably around silhouettes. We wish to avoid such discontinuities, and provide a smooth transition between regions of different convexity information: for instance, we wish to create a saddle at the T-junction between a ridge and a valley. This can be easily done by integrating principal curvatures in small neighborhoods in image-space: using a smooth integration kernel, we create transitions which vary smoothly in the shape descriptor domain. To this end, for a given pixel (x,y) in the image-plane, we apply a Gaussian blur to $k_1(x,y)$ and $k_2(x,y)$:

$$\begin{aligned} k_1^*(x,y) &= k_1(x,y) \otimes g(x,y,\sigma) \\ k_2^*(x,y) &= k_2(x,y) \otimes g(x,y,\sigma) \end{aligned}$$

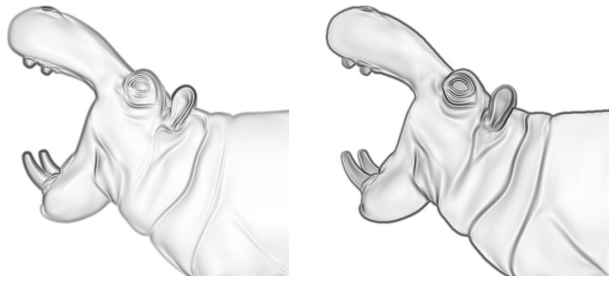
where σ is a scale parameter that controls the amount of blurring. Thanks to the separability of the Gaussian kernel ($g(x,y,\sigma) = g(x,\sigma)g(y,\sigma)$), the whole process can be efficiently performed on modern graphics hardware using two rendering passes in pixel shaders. Given a pair of smooth curvatures k_1^* and k_2^*, we compute $\vec{\mathbf{D}}$, and then proceed to the extraction of $\mathbf{L}$ in image-space.

3.2 Image-space information

We now need to determine the view-dependent curvedness information of our shape descriptor. Another goal is to provide automatic LODs, so that when the object gets closer or farther from the point of view, the shape descriptor will reveal finer or coarser shape cues respectively. Our approach is to compute curvedness as the amount of variation across normals in a pixel neighborhood. An important observation is that the variation of normals cannot be accurately computed by only using a scalar function. For instance, one could imagine computing derivatives of surface *slant* for this purpose, defined as the dot product between the normal and view vector. Unfortunately, this will miss many salient shape features such as the front facing edge of a cube for instance (see Figure 4 for a more complex example).

The intuition behind our solution is thus to compute curvedness information *both* along the $\mathbf{x}$ and $\mathbf{y}$ axis of the picture plane, and combine them to get $\mathbf{L}$, so that the most prominent shape features are identified in the end. More precisely, we are interested in the variation in normals along each of the picture plane dimensions, i.e, derivatives of the two first coordinates n_1 and n_2 of normals after they have been transformed to camera space. As our goal is to extract continuous curvedness information, we compute such a derivative not by considering an infinitesimal neighborhood, but by differentiating in an extended neighborhood. To this end, we convolve each of the normal coordinate images with Gaussian derivatives:

$$\begin{aligned} \nabla n_1(x,y) &= \begin{pmatrix} n_1(x,y) \otimes g_x(x,y,\sigma) \\ n_1(x,y) \otimes g_y(x,y,\sigma) \end{pmatrix} \\ \nabla n_2(x,y) &= \begin{pmatrix} n_2(x,y) \otimes g_x(x,y,\sigma) \\ n_2(x,y) \otimes g_y(x,y,\sigma) \end{pmatrix} \end{aligned}$$



(a) (b)

Figure 4: ***Comparison of curvedness computations****: (a) only using the gradient of surface slant for curvedness misses many salient shape features. (b) In contrast, our curvedness attribute takes into account the entire set of shape features.*

where $g_x(x,y,\sigma)$ and $g_y(x,y,\sigma)$ correspond to Gaussian derivative kernels in the **x** and **y** directions respectively, and σ determines the scale at which differentiation is performed (see [ter Haar Romeny 2003] for further details). In practice, we again make use of the separability of the Gaussian derivative operator ($g_x(x,y,\sigma) = g'(x,\sigma)g(y,\sigma)$ and $g_y(x,y,\sigma) = g(x,\sigma)g'(y,\sigma)$) as in Section 3.1.

The last step needed to obtain **L** from normal variations is to combine ∇n_1 and ∇n_2. We take inspiration from previous work that faced the general problem of computing the gradient of a multi-valued image. In [Zenzo 1986], a general solution is presented, that has been used for instance to compute color gradients from $(\nabla R, \nabla G, \nabla B)$ color triplets. We apply it to our normal variations $(\nabla n_1, \nabla n_1)$, and describe the process in the following. The method first computes directional gradients in vector space:

$$\nabla n_x = \begin{pmatrix} \nabla n_1 \cdot \mathbf{x} \\ \nabla n_2 \cdot \mathbf{x} \end{pmatrix} \quad \nabla n_y = \begin{pmatrix} \nabla n_1 \cdot \mathbf{y} \\ \nabla n_2 \cdot \mathbf{y} \end{pmatrix}$$

Then, it builds the symmetric tensor field N defined by

$$N = \begin{pmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{pmatrix} = \begin{pmatrix} \nabla n_x \cdot \nabla n_x & \nabla n_x \cdot \nabla n_y \\ \nabla n_y \cdot \nabla n_x & \nabla n_y \cdot \nabla n_y \end{pmatrix}$$

where $\cdot$ denotes the dot product. In [Zenzo 1986], it is shown that the combined gradient corresponds to the maximum absolute eigenvalue of N. We thus perform a Principal Composant Analysis and set **L** equal to the computed maximum absolute eigenvalue. The resulting curvedness attribute identifies the most salient shape features of the depicted object, including the ones missed using a derivative of surface slant (see Figure 4 for a comparison).

3.3 Combining measures

In previous sections, we explained how to extract $\vec{\mathbf{D}}$ and **L** at a given scale σ. The reason for using the same scale for both measures is to ensure that the convexity information revealed in smoothly varying curvedness areas is similarly smooth. We must also take care when $||\vec{K}|| = 0$ causing $\vec{\mathbf{D}}$ to be undefined. This is not an issue as long as $\mathbf{L} = 0$, therefore we impose it as a constraint wherever $\vec{\mathbf{D}}$ is undefined. In practice we use a smooth transition from **L** to $||\vec{K}||$ near $||\vec{K}|| = 0$ to avoid discontinuities. The resulting vector $\vec{\mathbf{S}} = \vec{\mathbf{D}} \cdot \mathbf{L}$ is our Apparent Relief Descriptor (see Figure 3).

In addition, the scale parameter may be used to control the spatial smoothness of our descriptor. We provide two different approaches: it may be set globally for the whole image, and controlled by the user; or it may be varied spatially per pixel using an arbitrary function such as depth or a point of focus around the mouse (see Figure 5). To implement the spatially-varying approach, one needs to compute **L** at multiple scales and combine them linearly, which is easily done on modern graphics hardware using multiple passes and still runs in real-time. Finally, we also allow the user to control **L** using the emphasis function $\mathscr{E}_\beta(x)$ described in the Appendix.

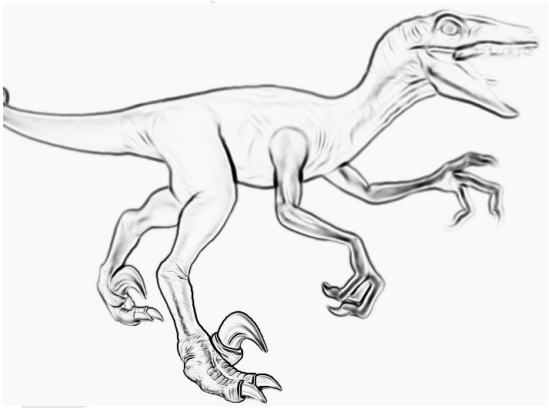


Figure 5: ***Levels-of-detail effects****: Varying scale per pixel provides a way to easily create LODs. In this example, we focus on the raptor's front leg, but any spatial function may actually be used.*

4 Shading styles

To illustrate the benefits of our Apparent Relief Descriptor, we explain in detail how it is used to depict shape with four different shading styles: cel shading, cartoon shading, minimal shading and exaggerated shading. The overall approach is to create a texture that we call an Apparent Relief Map (ARM), which assigns a *relief value* $r \in [0,1]$ to every possible shape vector. For a given pixel, we then simply use $\vec{\mathbf{S}}$ as a texture coordinate to lookup a relief value in the ARM. This texture is a straightforward way to choose which shape cues are selected. An example is given in Figure 2(b): here, everything but planar- and saddle-like surface regions is selected. All the textures in this paper have been done by hand in a conventional image processing software.

All our examples run in real-time on modern graphics hardware. Mesh sizes and framerates are given in figure captions, and are similar for most styles, except for exaggerated shading that requires more computations.

Cel-shading (Figure 6) is the simplest of the four shading styles we present. The idea is to make direct use of relief values from a given ARM as pixel colors. Because this style clearly shows the effect of the chosen ARM, we illustrate it using four different textures. The top-most one simply filters out every surface region that is planar. The ones below provide a finer control over which shape cues should be conveyed. The resulting shadings give compelling contour renderings.

Conventional **cartoon-shading** [Decaudin 1996] may easily be modified to take into account our shape descriptor (see Figure 7). The easiest approach is to employ the X-Toon shader of Barla et al. [2006]: the classic 1D toon texture is then extended to a 2D toon texture, where the vertical axis corresponds, in our case, to relief values. To illustrate this shading style, we use the ARM of Figure 6(c). As shown in Figure 7, X-toon allows a user to create more complex shadings, such as smooth color variations located only in dark regions, and removal of shape cues done by trimming the alpha channel around strong relief values.

Another place where a shape descriptor is needed is when using drastic shading styles, such as with what we call **minimal-shading**. A good example of this shading style is the appearance of the recent feature-length movie "Renaissance" [Volckman 2006]. Mostly black and white colors are used, so that artists often need to carefully tweak light positions by hand to reveal objects' shape. This is because shape cues are not always visible under conventional lighting, and they need to be re-introduced in some way, an unintuitive and time consuming process. The recent paper of DeCarlo et al. [DeCarlo and Rusinkiewicz 2007] proposes to extract highlight lines for reintroducing sharp shape cues in dark regions. With our shape descriptor, we rather reintroduce continuous shape cues in such a black and white shading style (see Figure 8): the idea is to treat relief values obtained from the ARM as contrast values. Minimal shading is thus defined by a simple linear interpolation between a diffuse and inverted diffuse intensity, using relief as the interpolation parameter: $I = (1-r)\mathbf{n}\cdot\mathbf{l} + r(1-\mathbf{n}\cdot\mathbf{l})$, where n and l define the current point's normal and light unit vectors. I is then thresholded at 0.5. To illustrate this style, we use an ARM that filters out everything but convex shape cues.

Our last example shows how to control **exaggerated shading** with our shape descriptor (see Figure 9). The main idea in [Rusinkiewicz et al. 2006] is to build a multi-scale analysis of object normals, in the spirit of a Gaussian Pyramid: at each level, normals are averaged to yield a coarser description. Then the authors apply a cosine shading at each level $i = 0..N$ using each time the light direction projected onto the plane defined by the normal in the coarser level. This has the effect of emphasizing local details at each level. Finally, the resulting lighting calculations are linearly combined using per-level weights k_i. We propose to use our apparent relief descriptor to accurately control the location of details that the user wishes to exaggerate. Our approach is to compute the weights according to relief values: $k_i = \mathscr{E}_\beta(r)$, where $\mathscr{E}_\beta$ is the emphasis function given in the Appendix, and $\beta \propto N - i/N$. Intuitively, this choice of weights progressively brings in finer and finer details with increasing relief. With our approach we can selectively enhance convex or concave cues as shown in Figure 9.

Many other ARMs and style parameters could be used, and we show additional results in the supplemental video. Moreover, the method may be applied to animated objects, provided principal curvatures are pre-computed at each vertex for every frame of the animation. The accompanying video also demonstrates our shading styles on such input as well as Figure 10, which shows frames of an animated 3D object rendered in a cartoon style.

5 Discussion and future work

We presented a new shape descriptor called Apparent Relief, that makes use of both object-space and image-space attributes to extract convexity and curvedness information respectively. It provides a flexible approach to the selection of continuous shape cues, and thus is efficiently used to depict shape through many different shading styles. On top of these benefits, the proposed method provides automatic LOD functionalities that enable a variety of new effects, works in real-time on modern graphics hardware, and is simple to implement.

To illustrate its potential, we demonstrated its use with four shading styles, including intuitive approaches to control minimal shading and exaggerated shading. Any salient shape cue identified via our descriptor is selected or filtered out via an Apparent Relief Map, and we believe this tool to be intuitive to manipulate. An even more direct control would be to provide a sketching interface to choose which cues are selected and which ones are filtered.

There are some limitations to our approach that we plan to address in future work. First, while we can deal with animated scenes, dynamic scenes (e.g. live character animations or natural phenomena) are not tractable because of the principal curvatures that need to be recomputed at each frame. Note that this is also a limitation of most object-based approaches. Second, while controlling the scale parameter enables many interesting LOD behaviors, it would be even more interesting to design an automatic scale selection mechanism. For this reason, we are interested in pursuing further the use of Scale Space Theory [ter Haar Romeny 2003] for the depiction of shape through shading.

References

BARLA, P., THOLLOT, J., AND MARKOSIAN, L. 2006. X-toon: An extended toon shader. In *NPAR*, ACM.

DECARLO, D., AND RUSINKIEWICZ, S. 2007. Highlight lines for conveying shape. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, ACM, 63–70.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Trans. Graph.(Proc. SIGGRAPH) 22*, 3, 848–855.

DECARLO, D., FINKELSTEIN, A., AND RUSINKIEWICZ, S. 2004. Interactive Rendering of Suggestive Contours with Temporal Coherence. In *NPAR2004*, ACM Press, A. Hertzmann and C. Kaplan, Eds., 15–24.

DECAUDIN, P. 1996. Cartoon looking rendering of 3D scenes. Research Report 2919, INRIA, June.

GOOCH, B., SLOAN, P.-P. J., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. 1999. Interactive technical illustration. In *I3D*, ACM, 31–38.

HERTZMANN, A. 1999. Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines. In *ACM SIGGRAPH 99 Course Notes. Course on Non-Photorelistic Rendering*, S. Green, Ed. ch. 7.

HOGARTH, B. 1991. *Dynalic light and shade*. Watson-Guptill Publications.

JEONG, K., NI, A., LEE, S., AND MARKOSIAN, L. 2005. Detail control in line drawings of 3D meshes. *The Visual Computer 21*, 8-10 (September), 698–706. Special Issue of Pacific Graphics 2005.

JUDD, T., DURAND, F., AND ADELSON, E. H. 2007. Apparent ridges for line drawing. *ACM Trans. Graph.(Proc. SIGGRAPH) 26*, 3, 19.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Trans. Graph. (Proc. SIGGRAPH) 22*, 3 (July), 856–861.

KOENDERINK, J. J., AND VAN DOORN, A. J. 1992. Surface shape and curvature scales. *Image Vision Comput. 10*, 8, 557–565.

LEE, C. H., AND HAO, X. 2006. Geometry-dependent lighting. *IEEE Transactions on Visualization and Computer Graphics 12*, 2, 197–207. Member-Amitabh Varshney.

MILLER, G. 1994. Efficient algorithms for local and global accessibility shading. In *ACM SIGGRAPH '94*, 319–326.

NI, A., JEONG, K., LEE, S., AND MARKOSIAN, L. 2005. Multiscale line drawings from 3D meshes. Tech. Rep. CSE-TR-510-05, Department of Electrical Engineering and Computer Science, University of Michigan, July.

NIENHAUS, M., AND DÖLLNER, J. 2005. Blueprint Rendering and Sketchy Drawings. In *GPU Gems II*, M. Pharr, Ed. Addison-Wesley Professional, 235–252.

OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.(Proc. SIGGRAPH) 23*, 3, 609–612.

RUSINKIEWICZ, S., BURNS, M., AND DECARLO, D. 2006. Exaggerated shading for depicting shape and detail. *ACM Trans. Graph (Proc. SIGGRAPH) 25*, 3 (July).

RUSINKIEWICZ, S. 2004. Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission*.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-d shapes. In *ACM SIGGRAPH '90*, 197–206.

SCHLICK, C. 1994. *Graphics Gems V*. Morgan Kauffman, ch. A Fast Alternative to Phong's Specular Model, 385–384.

TER HAAR ROMENY, B. 2003. *Front-End Vision and Multi-Scale Image Analysis*. Kluwer, August.

VOLCKMAN, C., 2006. Renaissance.

WOOD, P. 1994. *Scientific Illustration*. Wiley.

ZENZO, S. D. 1986. A note on the gradient of a multi-image. *CVGIP 33*, 1, 116–125.

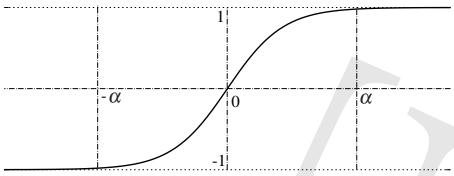
ZHUKOV, S., INOES, A., AND KRONIN, G. 1998. An ambient light illumination model. In *Rendering Techniques '98*, Springer-Verlag Wien New York, G. Drettakis and N. Max, Eds., Eurographics, 45–56.

Appendix

Scaling function: To remap curvatures from $]-\infty,\infty[$ to $]-1,1[$, we use a function based on *tanh*:

$$\mathscr{S}_\alpha(x) = tanh\left(x\frac{K}{\alpha}\right), \text{ with } K = tanh^{-1}\left(\frac{2^B-2}{2^B-1}\right).$$

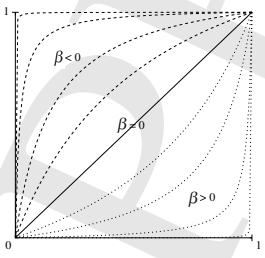
where B controls the precision in number of bits (we use $B=8$). This function remaps the range $[-\alpha,\alpha]$ to $]-1,1[$, since for each $|x|>\alpha$, $\mathscr{S}_\alpha(x)$ will be rounded to 1.



Emphasis function: Based on the rational function [Schlick 1994], we define $\mathscr{E}_\beta : [0,1] \to [0,1]$ to exaggerate or reduce local variations:

$$\mathscr{E}_\beta(x) = \frac{x}{exp(\beta)(1-x)+x}$$

This function is controlled via a parameter β, which gives intuitive variations for exaggeration ($\beta > 0$) as well as attenuation ($\beta < 0$).



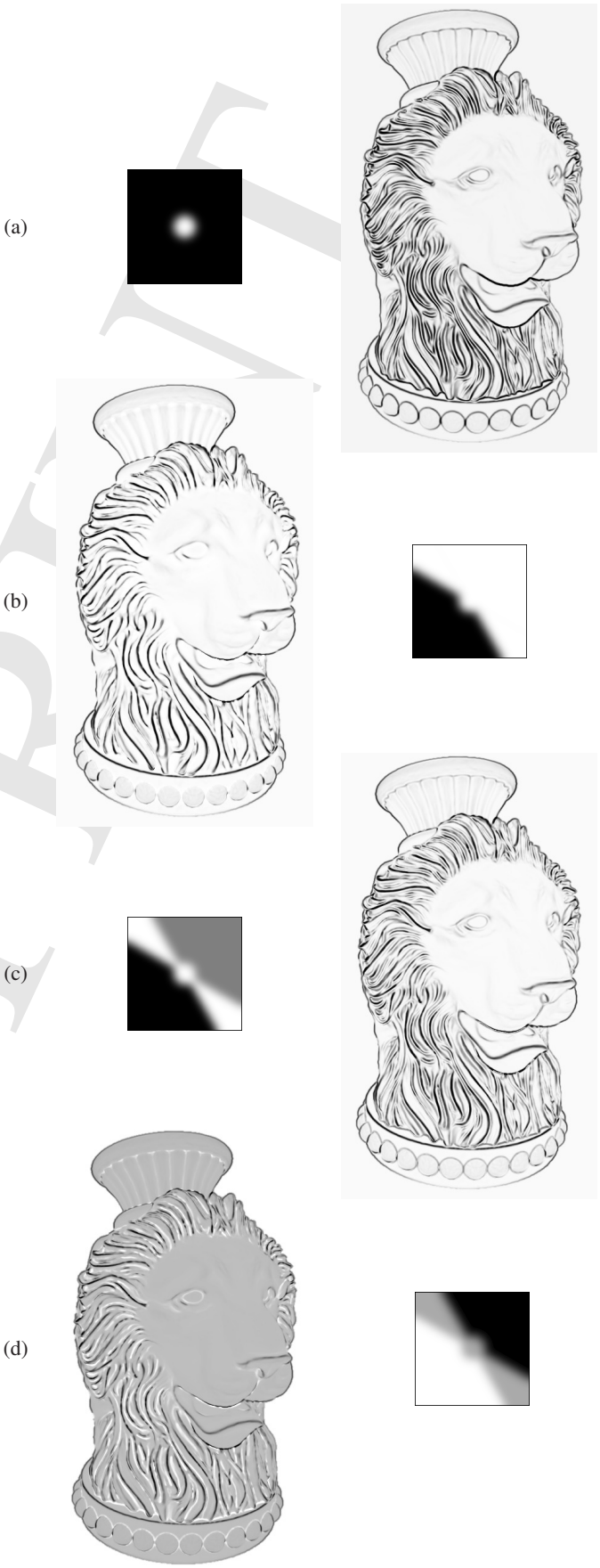
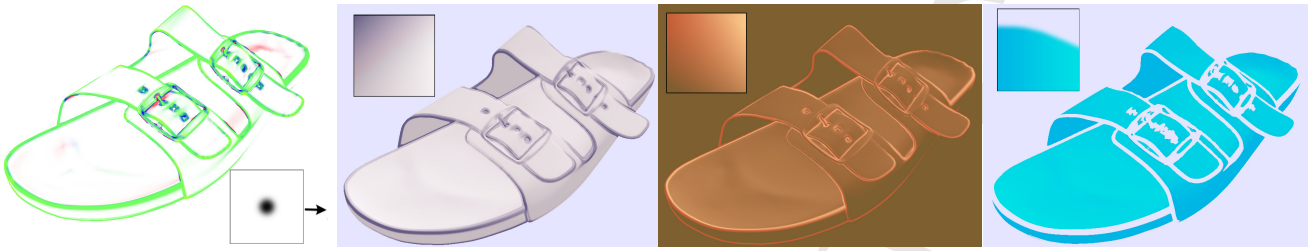


Figure 6: ***Cel-shading:*** *By simply displaying relief values, we show the versatility of using an ARM as a control interface. In (a), we filter only planar regions; in (b), we keep only convex cues. (c) shows an intermediate result between (a) and (b), where concave cues are displayed in light gray. As a last example, (d) renders concavities in white, convexities in black, and planar regions in gray (0.4 Mtri / 244 fps).*

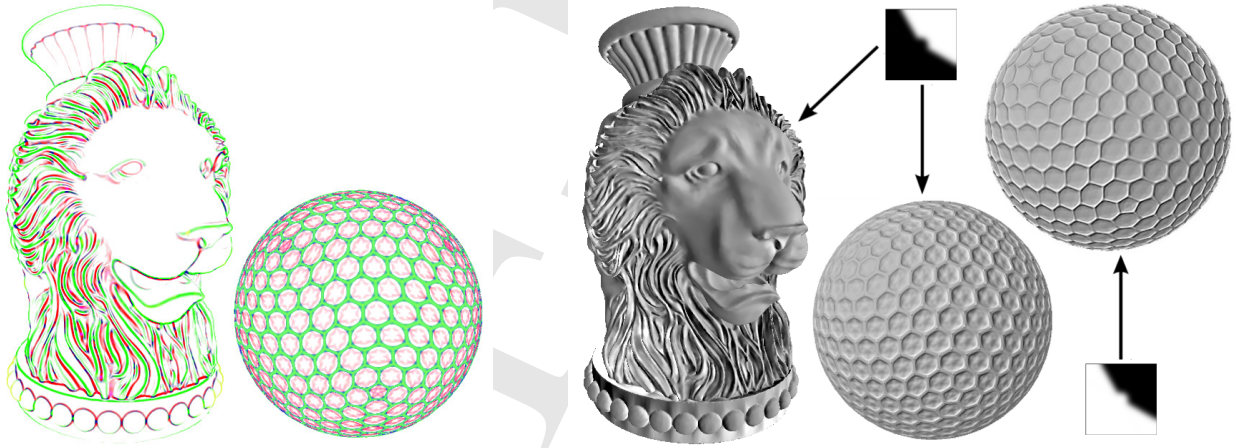


(a) (b) (c) (d)

Figure 7: ***Cartoon-shading:*** *The apparent relief descriptor (a) is used to select convex and concave regions and to lookup into X-Toon textures. In (b), we use a smooth color gradation that gradually fades out in lit regions. In (c), we reverse lights and darks, and show shading variations* inside *shape cues. In(d), we use a very subtle color gradient, and trim the alpha channel when the relief value is high, removing shape cues and flattening the result (0.3 Mtri / 250 fps).*

(a) (b)

Figure 8: ***Minimal-shading:*** *The apparent relief descriptor (a) allows to reintroduce the contrast lost using a drastically simple shading. Convex regions are used to make shape cues appear in black-on-white, or white-on-black: in (b), from left to right, we first rotate around the object to show its dark side, and then move the light around (0.15 Mtri / 250 fps).*



(a) (b)

Figure 9: ***Exaggerated-shading:*** *The apparent relief descriptor (a) may be used to control exaggerated shading. Relief values govern the amount of exaggerated details in specific regions: we use concave cues, except for the right-most image where convex cues are selected (1 Mtri / 40 fps).*

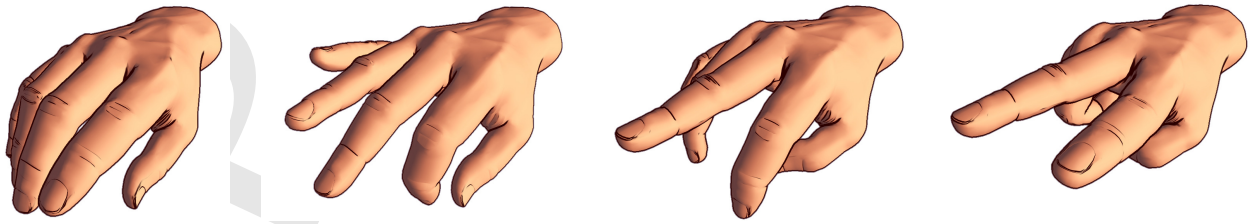


Figure 10: ***Animated example:*** *Four frames of an animated object, rendered using a cartoon shading style. Note the fine wrinkles that appear on fingers.*