X-Toon: An Extended Toon Shader

Pascal Barla, Joëlle Thollot
ARTIS GRAVIR/IMAG INRIA* Grenoble France

Lee Markosian
University of Michigan

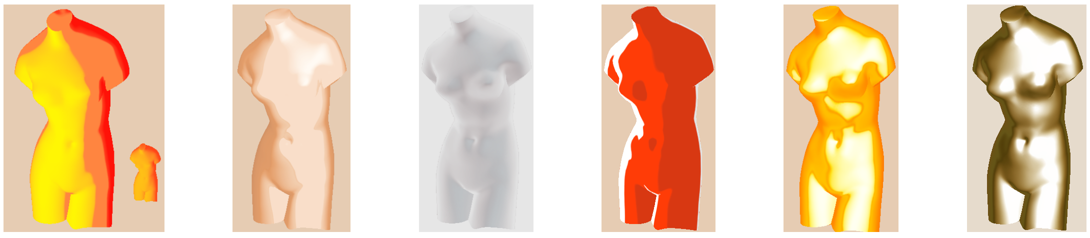


Figure 1: Some example effects achieved by our extended toon shader: continuous levels of abstraction (a); abstraction of near-silhouette regions: smoothing (b), and opacity (c); backlighting (d); and highlighting: plastic (e), and metal (f).

Abstract

Traditional toon shading uses a 1D texture that describes how tone varies with surface orientation relative to a given light source. In this paper we describe two extensions to the basic algorithm that support view-dependent effects. First, we replace the 1D texture with a 2D texture whose second dimension corresponds to the desired "tone detail," which can vary with depth or surface orientation. This supports effects such as levels-of-abstraction, aerial perspective, depth-of-field, backlighting, and specular highlights.

Second, we further extend the toon shader to use a modified normal field that can range from the original normals to a simpler set of normals taken from an "abstracted shape." A global shape detail parameter determines the degree of interpolation between the original and abstracted normal fields. We explain how to implement these ideas efficiently on the GPU via vertex and fragment shaders, and discuss ways to extend our approach to alternative tone and shape detail maps.

1 Introduction

Over the past decade, "toon" shading has proven popular in a variety of 3D renderers, video games, and animations. The idea is simple but effective: extend the lambertian shading model by using the computed illumination (a dot product between a light vector and the surface normal) to index into a 1D texture that describes how the final shading varies from "dark" to "light" regions. The designer controls the behavior of the shader by creating a 1D texture, typically composed of two or three regions of constant color, to mimic the flat regions of constant colors found in comics 漫画 and traditional animation. Toon shading can be implemented efficiently via vertex and fragment programs on modern GPUs.

On the other hand, toon shading does not reflect the desired level of abstraction (LOA) of a surface. Such LOA behavior plays an important role in traditional media, however. Often, some objects are considered less important (e.g., characters in the background) and thus are depicted with greater abstraction [McCloud 1994]. In paintings and drawings, an effect known as aerial perspective makes objects in the background appear desaturated and less detailed than those in the foreground. And in scientific illustration, a technique similar to depth-of-field is used to focus on a region of a shape by decreasing contrast or opacity in less-important or "out of focus" parts of the surface [Wood 1994].

Conventional toon shading is view-independent, and so cannot represent plastic or metallic materials, for which view-dependent highlights are of primary importance. Similarly, it cannot support the view-dependent backlighting effects, often used in traditional comics and animation, in which a virtual "back light" illuminates the surface near the silhouette.

Finally, in conventional toon shading, every surface location is rendered with full accuracy, so that even small shape features are depicted by the shading (at least for some light directions). This can be desirable, but often designers working traditionally apply a degree of abstraction so that small shape features are omitted. A similar ability to depict an abstracted version of the shape is thus desirable in an automatic toon shader.

In this paper we describe X-Toon, a toon shader that supports view-dependent effects through two independent extensions to conventional toon shading. The first incorporates a notion of **tone detail**, so that tone varies with depth or orientation relative to the camera. For this, we replace the conventional 1D texture used in toon shading with a 2D texture, where the second dimension corresponds to desired tone "detail" (see below). We describe several ways to define the additional texture coordinate.

Our second extension, presented in Section 4, lets us vary the perceived **shape detail** of the shading. We achieve this by using a modified normal field defined by interpolating between normals of the original shape and normals of a highly abstracted shape. This approach has the advantage of abstracting the shading from a shape (while preserving silhouettes).

Note that "detail" here corresponds to visual abstraction: less detail means greater abstraction. We use the term LOA instead of LOD to emphasize that our goal is visual abstraction, not increased rendering speed (the usual motivation for LODs in computer graphics).

The main contribution of this paper is a simple, unified approach that lets a designer intuitively achieve a variety of effects, including those that rely on *correlating* desired tone detail with the underlying tone value, as explained in Section 3.

*ARTIS is a research team of the GRAVIR/IMAG laboratory, a joint unit of CNRS, INPG, INRIA and UJF.

2 Related Work

Toon shading is available in production software such as Renderman, Softimage, Maya, and Autodesk 3ds Max. It has been used in many video games, TV shows, and feature films. The first published description of the method is due to Lake *et al.* [2000].

Related methods include the technical illustration shader of Gooch *et al.* [1998] and the "Lit Sphere" of Sloan *et al.* [2001]. The latter uses a painted spherical environment map instead of a 1D texture. Neither handles the view-dependent behaviors we are targeting.

Mip-maps provide one means of taking depth into account as an attribute. Klein *et al.* [2000] used specially constructed mip-maps (called "art maps") to achieve constant-sized strokes in textures applied to 3D scenes. The "tonal art maps" of Praun *et al.* [2001] extended this approach to take lighting into account in hatching patterns.

We could similarly define specialized art maps for the management of 1D toon textures with LOAs. Instead, we prefer a more general approach whereby a 2D texture represents an ordinary toon texture at a continuous range of abstraction represented by the added dimension. This way, the LOA selection mechanism can be more general than the fixed depth-based computations used in mip maps. It also lets designers create LOA toon textures directly as 2D images, e.g. by painting them in a paint program. The only requirement is to understand that the 2 dimensions of the texture correspond to tone and LOA, respectively. This approach retains the simplicity of the classic toon shader and does not constrain the designer with a set of predefined behaviors and discrete LOAs. We compare our approach with a 1D mip-map approach in the accompanying video.

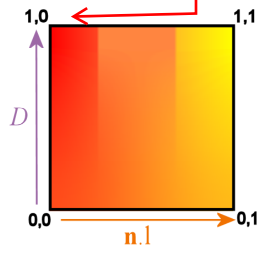
Anjyo *et al.* [2003] described techniques to add stylized highlights to a classic toon shader, thus taking viewing orientation into account. Their technique, which acts on top of a toon shader, supports abstraction of highlights through the use of translation, rotation, splitting and squaring operations. Other methods could be devised to control the highlights independently of the tone computation (e.g., using a 1D highlight texture). However, once the properties of a highlight (e.g., size, softness, color) are chosen, they remain the same under any viewing condition. Thus, the method of Anjyo *et al.* only considers highlights with hard contours, a single color and a fixed size. Our approach offers the ability to correlate properties of the highlights with the tone value, in a way precomputing the response of the highlight to changing viewing conditions: highlights can change in softness, color and size dynamically. We compare our approach with an independent specification of highlights and tone in the accompanying video.

Anjyo's method does permit modifications to the shape of the highlight, thus controlling what we term its "shape detail." Achieving such an effect in a toon shader is an interesting avenue for future work (see Section 5).

Other methods manipulate the amount of shape detail conveyed in non-photorealistic renderings. The "graftal textures" of Markosian *et al.* [2000] manage small detail elements ("graftals") representing leaves, hair, etc., introducing and removing them according to a desired level of detail. Deussen and Strothotte [2000] do something similar for 3D models of trees rendered in a pen-and-ink style. The designer can choose the desired level of abstraction via a slider. Our goal is to have similar capabilities in a toon-shader.

3 Tone detail

可以看成1D texture 的stack



We extend the classic 1D toon texture by adding a vertical "detail" axis (corresponding to LOA) to build a 2D toon texture. The horizontal axis corresponds to tone as in a classic toon texture – the texture coordinate along that axis is derived from a standard lambertian shading computation: $\mathbf{n} \cdot \ell$, where $\mathbf{n}$ is the unit surface normal and ℓ is the unit light direction. The vertical axis corresponds to tone detail: each value along this axis (labelled D for detail) corresponds to its own 1D toon texture. The whole 2D texture can thus be regarded as a stack of 1D toon textures with increasing "detail." The designer is free to create this 2D texture using any image processing or painting tools available. We generally found it convenient to start with a standard 1D toon texture at $D = 1$ and apply image processing transformations down the D axis to account for detail loss. In this paper and accompanying video we provide many example textures that have been created with this approach, and we demonstrate the resulting behavior of the extended toon shader.

Once the 2D texture is defined, the designer must choose an attribute (e.g., orientation or depth) that will control the tone detail, and provide functions called **attribute maps** that map the attribute to a detail value D at each location on a surface. This formulation is general in that any attribute can be chosen, depending on the application goals.

In this paper, we consider view-dependent attributes. In the next two sections we describe how to compute depth-based and orientation-based attribute maps to achieve LOAs, aerial perspective, depth-of-field, backlighting, and specular highlights.

3.1 Depth-based attribute mapping

We consider two ways to define the "depth" of a point in 3D: we can use its euclidean distance to the viewpoint, or its distance along the focal axis (see Figure 2). The latter assigns the same depth to all the points that lie in a plane parallel to the image plane. Depending on the intended effect, one computation or the other may be preferred. For LOAs and aerial perspective effects, we use distance along the focal axis; for depth-of-field effects, we prefer distance to the eye.

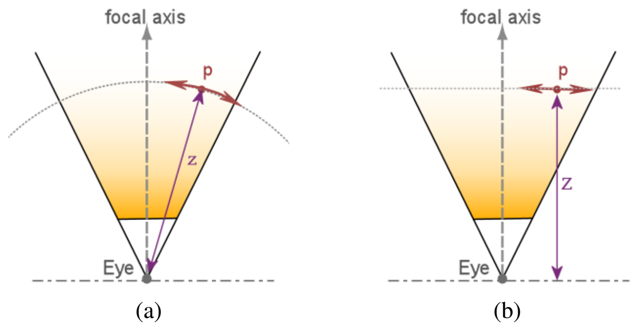


Figure 2: The depth of a point p relative to the eye can be calculated in two ways: (a) The distance between the eye and p or (b) the distance along the focal axis.

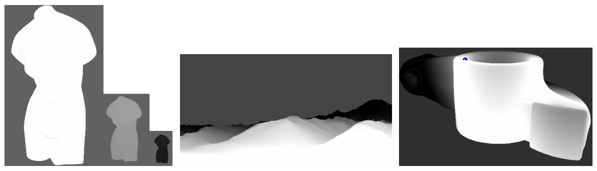


Figure 3: (a) The LOA attribute map used for the venus model, (b) the aerial perspective attribute map used to render the landscape model, and (c) the depth-of-field attribute map and the focus point (blue sphere) used to render the mechanical part model.

We derive a detail value $D \in [0..1]$ from the computed depth z via two user-specified parameters: z_{min}, the distance at which the detail starts decreasing, and $r > 1$, the scale factor that defines the coarsest detail (greatest abstraction) at distance $z_{max} = rz_{min}$. To account for perspective projection, we use the following formula for detail mapping[1]:

$$D = 1 - \log(z/z_{\min})/\log(z_{\max}/z_{\min})$$

$$D = 1 - \log_r(z/z_{\min})$$

For depth-of-field, we use a different formula for the detail computation: given a focus point c in 3D, we compute D as follows[2]:

$$D = \begin{cases} 1 - \log(z/z^-_{\min})/\log(z^-_{\max}/z^-_{\min}), & z < z_c \\ \log(z/z^+_{\max})/\log(z^+_{\min}/z^+_{\max}), & z > z_c \end{cases}$$

with $z^{\pm}_{\min} = z_c \pm z_{\min}$ and $z^{\pm}_{\max} = z_c \pm rz_{\min}$.

Depth-based attribute maps are shown in Figure 3. In our implementation, we compute D per vertex in a vertex shader. A pixel shader performs the 2D toon texture lookup using the value of D interpolated at each pixel. In an interactive session, the designer can thus experiment with the z_{min} and r parameters and set the point of focus, and then observe the resulting behavior of the X-Toon shader in real time.

Figure 7 shows depth-based tone LOA effects using 2D textures created in three different ways: in (b) we progressively blurred an input 1D toon texture, in (c) we interpolated between three 1D toon textures with smooth transitions, and in (d) we shifted and lightened a 1D texture to create a "receding shadows" effect. The LOA attribute map is shown in Figure 3(a).

Figure 8 shows two examples of a focus-based detail map. In (a), we used a smooth two-tone texture that converges to a constant color with detail loss. In (b), we use a texture that decreases opacity and contrast with detail loss, and only shadows and highlights are fully opaque. We used these effects to focus attention on a specific depth range of a mechanical model. The depth-of-field attribute map is shown in Figure 3(c), along with the focus point depicted by a small blue sphere.

空中透视效果

Finally, Figure 9 shows a landscape rendered with four different X-Toon implementations of aerial perspective. We created these textures using filters such as decreasing contrast (b), converging to a specific hue (c) and (e), and decreasing opacity (f). We did this independently for the main tone values (dark, intermediate, and highlights) so that we can correlate the change made by aerial perspective with a specific tone value (e.g., we make the intermediate colors more transparent with distance). The aerial perspective attribute map is shown in Figure 3(b).

[1]We assume $0 < z_{min} \leq z \leq z_{max}$.
[2]We assume $0 < z^-_{max} < z^-_{min} < c < z^+_{min} < z^+_{max}$.

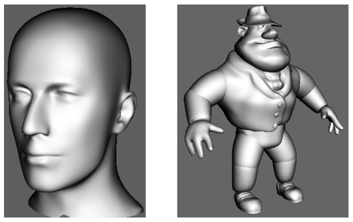


Figure 4: (a) The near-silhouette attribute map used for the face model. (b) The highlight attribute map used for the "Al" model. For these images, the detail scale is inverted, so that dark tones correspond to low detail values. These maps behave differently when the camera or lights are animated.

3.2 Orientation-based attribute mapping

We next consider an attribute map based on the orientation of the surface with respect to the observer. With the appropriate choice of 2D toon texture and policy for computing D, we can achieve effects such as fading of near-silhouette regions, or brightening and coloring of near-silhouette regions to suggest a virtual "backlight." We define the near-silhouette attribute mapping as follows: $D = |\mathbf{n} \cdot \mathbf{v}|^r$, where $\mathbf{n}$ and $\mathbf{v}$ are the unit normal and view vector, respectively, and $r \geq 0$ is a user-defined parameter that controls the magnitude of the effect.

Specular highlights are another view-dependent effect: these depend on the angle between the viewing direction and the light reflection vector, and the specular properties of the material. Depending on the chosen 2D texture, we can model various material highlights intuitively (e.g., plastics or metals). This is better than just compositing a specular layer over the ordinary 1D toon shader, because the 2D texture lets us control the profile of the highlight (smoothness, width, alpha) so that it is correlated with the underlying tone. In our implementation, we use the Phong highlight model to map the highlight attribute values to detail: $D = |\mathbf{v} \cdot \mathbf{r}|^s$, where $\mathbf{r}$ is the light reflection vector at the current surface location and $s \geq 1$ is the "shininess" coefficient set by the designer to control the magnitude of the effect. Orientation-based attribute maps are shown in Figure 4.

In practice, it is straightforward to compute these values in a vertex shader. For accuracy reasons, however, we pass the relevant vectors to a fragment shader where they are interpolated before being used in the detail computation.

Figure 10 shows how we can use the orientation detail map for different purposes. In (b) we use it with a texture similar to the LOA examples in Figure 7 to abstract tonal detail in near-silhouette regions. In (c), we make the surface fade out in near-silhouette regions to yield a "fuzzy" appearance. A different 2D texture is used in (d) to produce an effect we call "backlighting". With the appropriate choice of 2D texture, the thickness and color of backlit regions can be made to depend on the tone value of the underlying 1D toon texture. The near-silhouette attribute map is shown in Figure 4(a).

Figure 11 shows three types of highlights. The first depicts a plastic-like shader, where we control the thickness and opacity of the highlight directly in the texture. The second shows a metallic-like shader, where the highlight has a quality of "glowing" suddenly when the camera moves to a favorable orientation. While the first supports dynamic variation in the size of the highlights, the second controls the smoothness of tone transitions view-dependently. A third example modifies the colors that appear near transitions to highlights. The highlight attribute map is shown in Figure 4(b).

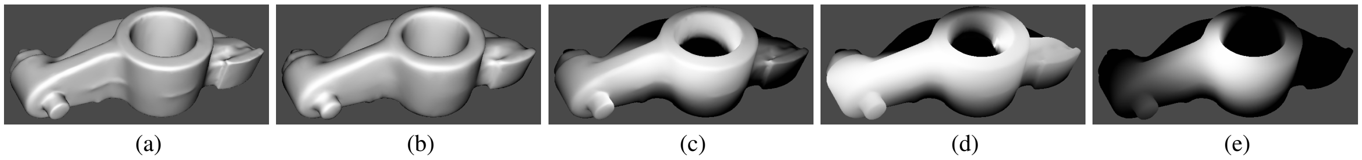


Figure 5: For each figure, darker tones correspond to normals that are more tilted away from the camera. (a) The normals of the input mesh. (b) Smoothed normals (note the shape details that disappeared in the front of the object). (c) Elliptic normals whose distribution follows the principal direction of elongation of the object. (d) Cylindric normals, oriented in the direction of principal elongation. (e) Spheric normals, highly abstracted from the original model. Note that for every geometric map, the silhouettes are preserved.

4 Shape detail

We now describe how to further modify the shading to depict an abstracted version of the shape. Note that this extension is independent of the tone detail extension and is done view-independently. We achieve it by modifying the surface normal field used in shading computations with a **geometric map**: we map the input mesh to an abstracted shape that acts as a lower bound for shape detail. We ensure this abstracted shape has a direct correspondence with the input surface. The designer controls the "abstraction" of shape detail by setting a blending parameter that controls how the shader interpolates between normals of the input mesh and the abstracted shape. Note that while normal vectors used in shading calculations are modified, vertex locations are not. This has the important property of ensuring consistency when combined with other rendering passes.

We describe four types of abstracted shape: a smoothed version of the original mesh, and an enclosing ellipse, cylinder, and sphere. We show examples of these abstracted shapes in Figure 5, rendered using a simple lambertian shader with the light at the camera to better visualize the abstraction. The dimensions of the ellipse come from a bounding box of the input mesh. To map normals from the input mesh to the sphere, cylinder, or ellipse, we use straightforward analytic mapping techniques. We create the smoothed mesh using a simple normal smoothing technique, and the 1-1 correspondence between mesh vertices defines the mapping. These four types of abstracted shapes provide a great deal of expressiveness: they let us "flatten" the shading in a way that resembles some drawings, paintings and comics. Note that this manipulation of normals to abstract the shading need not be restricted to toon shading.

In our implementation, we precompute and store abstracted normals at each mesh vertex. A vertex shader is used to linearly blend between the input and abstracted normals using a single (global) weight provided by the designer. The resulting vector is renormalized in the pixel shader where the toon texture lookup is performed. While simple, this method works reasonably well, is easy to implement, requires just one extra normal per vertex, and provides interactive feedback when implemented on the GPU. Of course, for the shading to be coherent, the normals of the highly abstracted shape should not exhibit any degeneracies; however we consider this issue to be beyond the scope of this paper.

5 Discussion and future work

We have presented two independent extensions to the original toon shader, aimed at retaining its simplicity while allowing more general behaviors. An important property of the original toon shader is that it is fast. To compare our approach in terms of efficiency, we have made a set of measurements summarized in Table 1. The rendering time of X-Toon for typical scenes is only about 20% slower than the original toon shader, and can thus be used in interactive applications as well as for off-line rendering.

Model	*Size*	*Resolution*	*Toon*	*X-Toon*	*Ratio*
Mech part	$10,000\ \circ$,	640*x*480	287	241	1.19
	$20,000\ \triangle$	1280*x*950	268	222	1.21
David	$26,051\ \circ$,	640*x*480	94	80	1.18
	$49,998\ \triangle$	1280*x*950	90	76	1.18
Terrain	$105,152\ \circ$,	640*x*480	33	28	1.18
	$208,962\ \triangle$	1280*x*950	32	27	1.19

Table 1: Run-time performance (in frames per second) and ratio of toon to X-Toon frame rates for three different models ($\circ$ = vertices, $\triangle$ = triangles) at two different resolutions. These measurements were made on a Pentium 4 with ATI Radeon 9800 GPU.

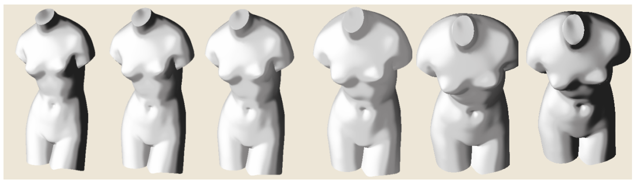


Figure 6: We show six frames of a small animation where coarser detail is automatically selected where the magnitude of optical flow is higher. We used the receding shadow texture from Figure 7.

Although we have focused in this paper on view-dependent attributes, our system can easily be extended to handle other attributes. For example, we can use optical flow to assign less detail to objects as they move faster in image space. To implement this method, we store the world-to-pixel matrix from the previous frame and use it to compute a per vertex displacement vector, measured in pixels. We use the length of this vector to assign a value to D. (See Figure 6.) In future work, we plan to investigate additional attribute detail maps, e.g. based on haloing, labeled importance, points of interest, cast shadows or reflections, and find a way to combine them efficiently.

Our geometric mapping is similar to the one developed by Ni *et al.* [2006], except that they interpolate normals *and* positions using a *local* detail map, and they support the use of multiple shapes (not just two) representing distinct levels of detail. Applying similar ideas in an X-Toon shader is one possibility for future work. Another possibility is to take inspiration from Anjyo *et al.* [2003], and use light maps to control the shape of highlights. We could also consider ways to control detail parameters over time (via keyframing, e.g.) as was done in the same paper.

We have only considered a single global weight to interpolate between the original and abstracted normals. A more flexible approach would be to control shape detail using a locally varying weight. This varying weight could be provided by the designer via a texture painted on the surface, or through an attribute map similar to those we have described for controlling tone detail. Controlling shape and tone detail via the same attribute maps may make sense in terms of providing more consistency in the resulting shading.

6 Conclusions

We have presented two methods for controlling the level of abstraction in toon shading. Our extended toon shader lets the designer correlate a chosen view-dependent attribute (e.g. depth or orientation) with tone detail by means of a 2D toon texture. Through the use of more or less abstracted normals, one can further abstract the shading from the shape it represents. By combining these extensions, a designer can easily create a wide range of effects that were difficult or impossible to achieve with previous approaches.

Acknowledgements

This research was supported in part by the NSF (CCF-0447883), and by a EURODOC grant (Région Rhône-Alpes).

References

ANJYO, K., AND HIRAMITSU, K. 2003. Stylized highlights for cartoon rendering and animation. *IEEE Computer Graphics and Applications 23*, 4, 54–61.

DEUSSEN, O., AND STROTHOTTE, T. 2000. Computer-generated pen-and-ink illustration of trees. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 13–18.

GOOCH, A., GOOCH, B., SHIRLEY, P. S., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 447–452.

KLEIN, A. W., LI, W. W., KAZHDAN, M. M., CORREA, W. T., FINKELSTEIN, A., AND FUNKHOUSER, T. A. 2000. Non-photorealistic virtual environments. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 527–534.

LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. 2000. Stylized rendering techniques for scalable real-time 3D animation. In *NPAR 2000: First International Symposium on Non-Photorealistic Animation and Rendering*, 13–20.

MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., HOLDEN, L. S., NORTHRUP, J. D., AND HUGHES, J. F. 2000. Art-based rendering with continuous levels of detail. In *NPAR 2000: First International Symposium on Non-Photorealistic Animation and Rendering*, 59–66.

MCCLOUD, S. 1994. *Understanding Comics*. Harper.

NI, A., JEONG, K., LEE, S., AND MARKOSIAN, L. 2006. Multi-scale Line Drawings from 3D Meshes. In *2006 ACM Symposium on Interactive 3D Graphics and Games*.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 579–584.

SLOAN, P.-P., MARTIN, W., GOOCH, A., AND GOOCH, B. 2001. The lit sphere: A model for capturing NPR shading from art. In *Graphics Interface 2001*, 143–150.

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

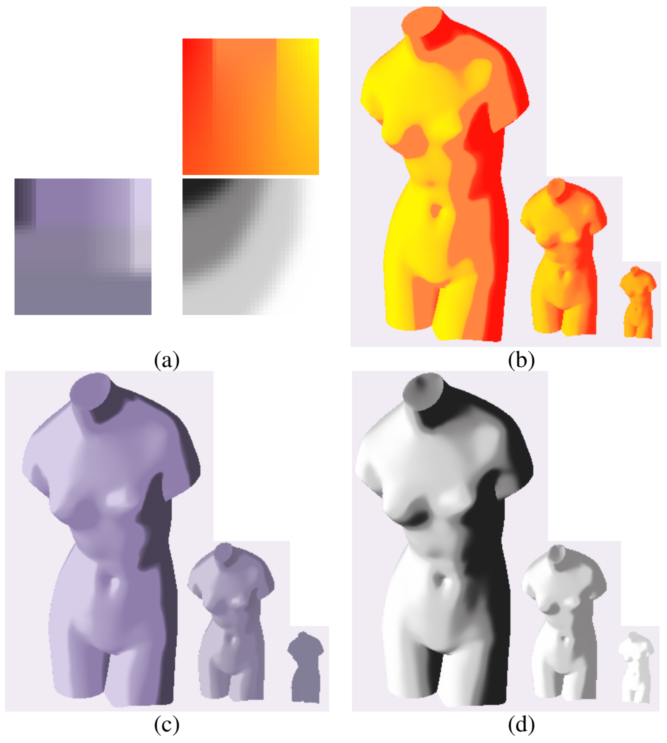


Figure 7: Some examples of the venus model rendered at various LOAs, with the corresponding toon textures in (a). In (b), we use a blurred texture to smooth out the shading with distance; in (c), we mimic a discrete LOA behavior using a texture with smooth steps along the detail axis, hence controlling the interpolation behavior; in (d), we make a receding shadow effect by sliding the darker tones to the left of the toon texture along the detail axis.

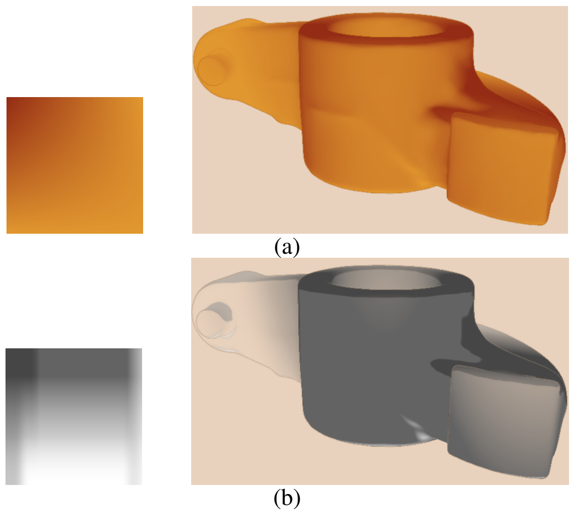


Figure 8: Some examples of depth-of-field shading: in (a), we use a blurred texture; in (b) we use a texture in which contrast and opacity are decreased for intermediate tones.

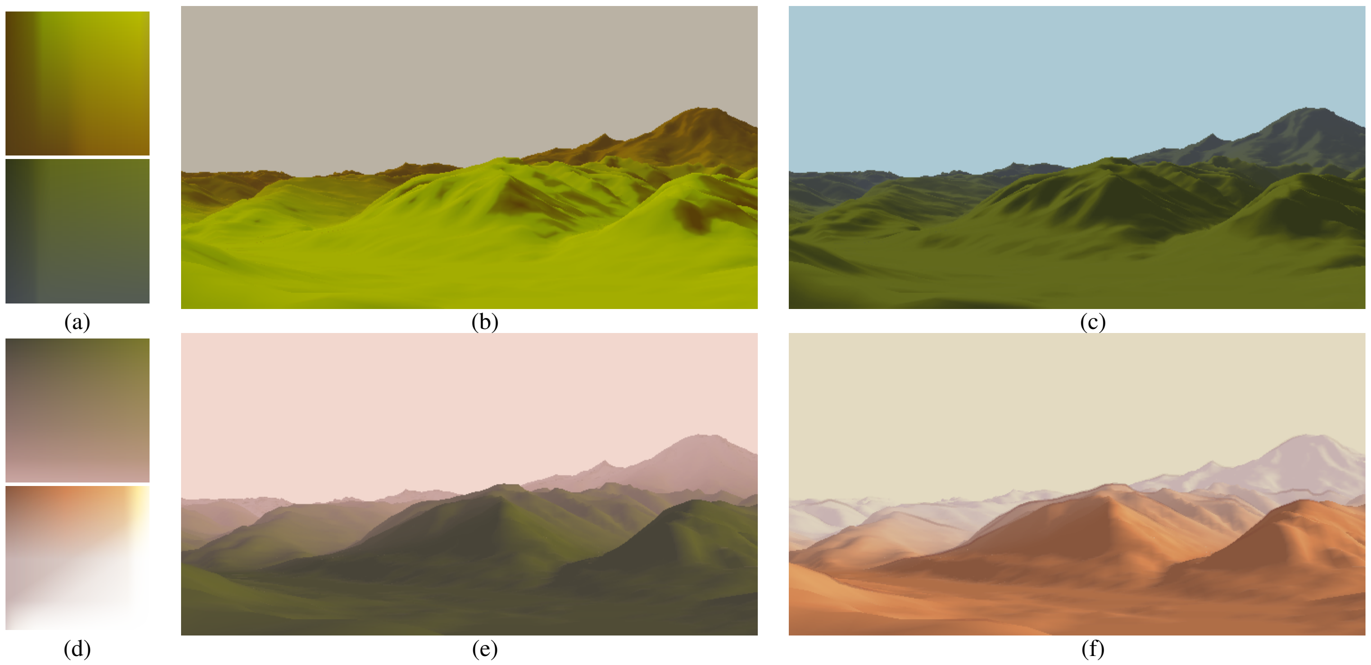


Figure 9: Aerial perspective effects. In (a) we show the two textures used to render the images in (b) and (c): the first one blends from a green-to-brown color ramp to a more uniform one consisting of brown tones; the second one applies a desaturation and shift toward blue. In (d) we show the two textures used to render the images in (e) and (f): the first one tends to a pink hue, with darker tones modified prior to lighter ones. The second texture decreases alpha and saturation, keeping only light gray shadows in the background.

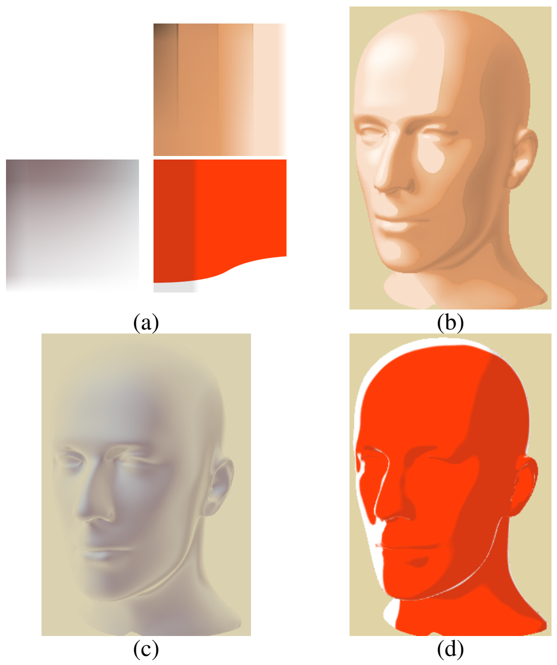


Figure 10: Near-silhouette abstraction and backlighting: (a) shows the textures used in (b), (c) and (d). In (b), we use a texture where intermediate tones are blurred prior to dark or light ones; this has the effect of smoothing out the shading in near-silhouette regions so that hard boundaries are only visible in regions facing the camera. In (c), the same approach is applied, this time to decrease opacity of the shading except for dark tones; this gives a fuzzy rendering of the model. In (d), we apply a simple white backlight that grows thinnner in darker tones; we can thus control the thickness of the backlighting by moving the light.

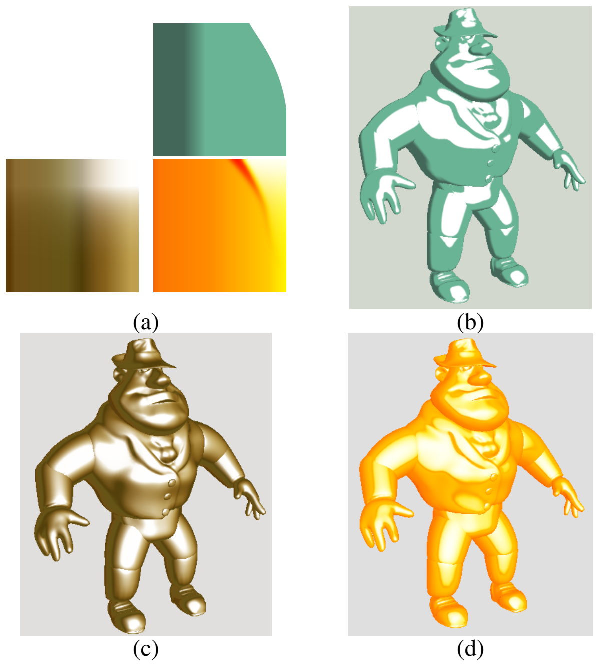


Figure 11: Plastic and metal highlights: (a) shows the textures used in (b), (c) and (d). In (b), we use a highlight texture that decreases in width, but is static with respect to color (white) and profile (hard boundaries); this gives a plastic toon highlight that varies in size depending on the viewing configuration. In (c), we use a texture that creates a "glowing" highlight, resembling a metallic material; this highlight is only present when the view direction is close to the reflected light direction, making the highlight appear suddenly, like a flare. In (d), we modify the color of the highlight, making it redder near its boundaries when it reaches a given scale.