Locally Controllable Stylized Shading

Hideki Todo* The University of Tokyo
Ken-ichi Anjyo† OLM Digital, Inc.
William Baxter‡ OLM Digital, Inc.
Takeo Igarashi§ The University of Tokyo

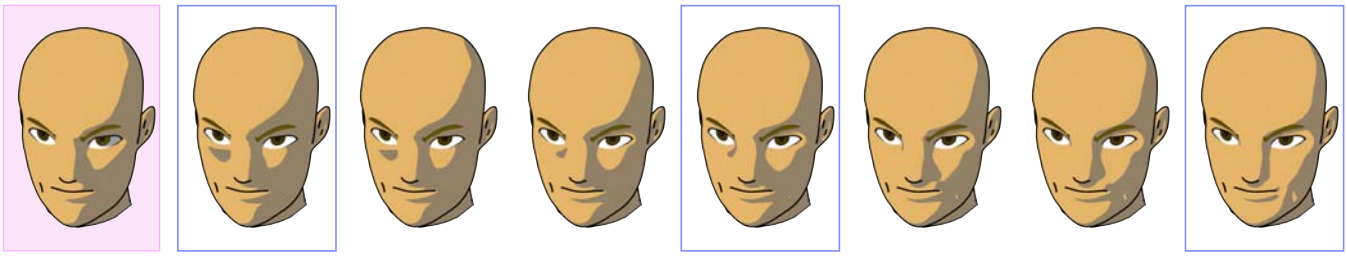


Figure 1: *Comparison of conventional toon shading* (leftmost image) *with our result* (remaining images). *Edits were made at the three key frames indicated:* (left) *added shaded area below left eye for expressive impact,* (middle) *deleted dark area around right eye, and* (right) *added shaded area below nose to emphasize three-dimensionality. These local edits integrate seamlessly with the global lighting, animate smoothly, and require no modification to the external lighting setup.*

Abstract

Recent progress in non-photorealistic rendering (NPR) has led to many stylized shading techniques that efficiently convey visual information about the objects depicted. Another crucial goal of NPR is to give artists simple and direct ways to express the abstract ideas born of their imaginations. In particular, the ability to add intentional, but often unrealistic, shading effects is indispensable for many applications. We propose a set of simple stylized shading algorithms that allow the user to freely add localized light and shade to a model in a manner that is consistent and seamlessly integrated with conventional lighting techniques. The algorithms provide an intuitive, direct manipulation method based on a paint-brush metaphor, to control and edit the light and shade locally as desired. Our prototype system demonstrates how our method can enhance both the quality and range of applicability of conventional stylized shading for offline animation and interactive applications.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.3.7 [Computer Graphics]: Animation

Keywords: non-photorealistic rendering, stylized shading, direct manipulation

*e-mail: td-rg7@ui.is.s.u-tokyo.ac.jp
†e-mail: anjyo@olm.co.jp
‡e-mail: baxter@olm.co.jp
§e-mail: takeo@acm.org

1 Introduction

We consider the problem of how to provide users with intuitive, fine-grained control over stylized light and shade on a 3D object. Over the past decade, a variety of non-photorealistic rendering techniques have been developed to facilitate visual interpretation of 3D objects. Most of these techniques are designed to elucidate particular attributes inherent to the object. For example, Gooch and Gooch [2001] developed a lighting model that changes hue to convey surface orientation, edge locations, and highlights for 3D technical illustration. The multi-scale shading method by [Rusinkiewicz et al. 2006] makes detailed 3D shape depiction at all frequencies possible.

On the other hand, in application fields such as digital animation and video games, there is a significant demand for locally controllable stylized light and shade, which can achieve results that are directable, intentional, and often fictive, yet ultimately more attractive for it. For example, the canonical cartoon shader used routinely in 3D animation often creates undesirable shaded areas. These can arise from the complexity of the underlying geometry or the complexity of the lighting, or just as a result of the basic physics of illumination. The left image in Figure 1 shows such an example, where the dark area partly covers the right eye of the character. Directors would like to have the ability have such features removed while retaining other dark areas. In other cases, they might like to request that a shaded area be added below the left eye, as shown in the second image from the left in Figure 1, in order to emphasize the character's fierceness. However, satisfying these diverse artistic requirements simultaneously would be very hard or almost impossible using only existing conventional lighting control and/or by fine-tuning the parameters used. Changing the geometry of the model or animating textures or light maps might be helpful for achieving this, but these are time-consuming and impractical on a production schedule. Despite the crucial importance of such fine-grained artistic control of stylized light and shade, very little research exists on how to provide such control or suitable interactive techniques to support it.

Our goal is to develop such "director-friendly" methodologies for stylistic depiction of light and shade. To explain our approach more concisely, we restrict the discussion for now to making 3D cartoon animation. In this case, due to the nature of stylistic depiction, the techniques used need not be physically realistic; however, they must

possess a certain sense of *plausibility* while meeting directorial demands. This emphasis on expressiveness over physical-realism implies that we must rely on the animator's creativity–more than automatic physically-based algorithms–to get a desired animation. Therefore, a stylized shading approach should provide a simple, intuitive user interface so that the animator can easily and interactively translate his or her creative vision into reality. A keyframe-based technique is appropriate, since it allows fine-tuning of stylistic animation in a traditional, but convenient and familiar way for animators. Additionally, real-time preview of the animation is also indispensable. These basic requirements for making stylized animation have led us to consider naïve key-framing as a first approach towards a new methodology. The overall process of the approach we propose is:

1. Begin by making an initial 3D scene, which includes the lighting and animation settings, using a conventional 3D software tool.

2. At each keyframe, the user designs and/or modifies the shaded area on a surface, using a paint-brush interface. This process is performed at interactive rates, prescribing the boundary constraint of the obtained area. Thereafter the new surface brightness distribution is automatically generated considering the boundary constraint.

3. The new surface brightness distributions at the keyframes are automatically transmitted to all the frames by linear interpolation. We thus obtain real-time preview of the stylistic animation.

The central idea of our approach is to effect the desired changes to light and shade boundaries by modifying the Lambertian $\mathbf{L}\cdot\mathbf{N}$ lighting term directly, adding a scalar offset function. This avoids the need to manipulate light vectors and normals and can be efficiently implemented using scalar-valued radial basis functions ([Wahba 1990]). The right images in Figure 1 are from an animation created using our techniques, while the leftmost shows the scene before modifications.

The rest of the paper is organized as follows. After briefly surveying related work in section 2, we describe the main ideas underlying the algorithms in section 3. In section 4, we describe some implementation details of our prototype system. Section 5 demonstrates animation examples and discusses our results. We conclude with some limitations and future work in section 6.

2 Related work

A number of NPR techniques, such as those in [Gooch and Gooch 2001], have been developed to emulate various stylistic appearances. For stylized rendering of 3D objects, Lake et al.[2000] proposed several fundamental real-time rendering techniques, including a traditional cartoon shader. The Lit-Sphere method by Sloan et al.[2001] can describe view-independent tone detail, using a painted spherical environment map. The WYSIWYG system by Kalnins et al.[2002] allows direct drawing of strokes onto 3D objects, while learning strokes by example. The multi-scale shading technique by Rusinkiewicz et al.[2006] can also control the appearance of shape detail by tuning parameters of the lighting model. Barla et al.[2006] proposed an extension of the traditional cartoon shader, which can control view-dependent tone detail, including such effects as aerial perspective and depth of field. The cartoon highlight shader in [Anjyo et al. 2006] allows a user to directly click-and-drag the highlights on a surface to design and animate them.

Previous work on user-specified indirect lighting design for photo-

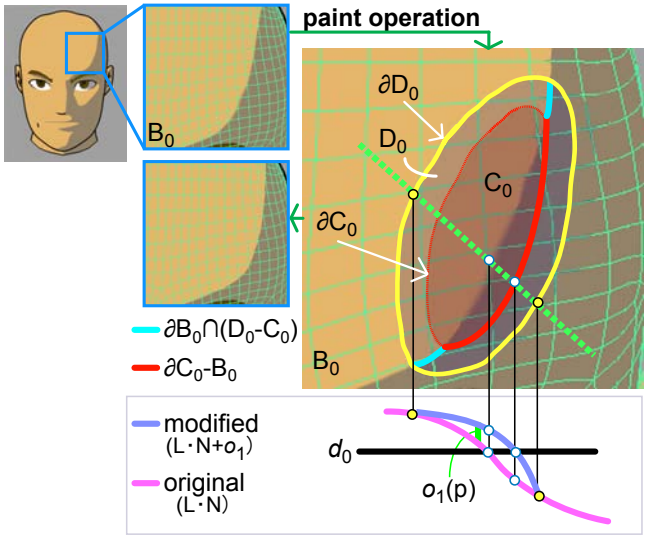


Figure 2: *Modifying a shaded area* $\mathbf{B}_0$ *with the paint brush interface: The resulting new area* $\mathbf{B}_0 \cup \mathbf{C}_0$ *can be represented functionally by introducing an offset function that modifies the standard* $\mathbf{L}\cdot\mathbf{N}$ *lighting term. The bottom graph shows a 1-d intensity distribution along the green line.*

realistic scene rendering is to some extent related to our approach as well. The design issue in photorealistic lighting is to find the light placement that results in the user-specified highlights and shadows in the scene (see [Lee et al. 2006] for more detailed discussion). There exist several good approaches ([Schoeneman et al. 1993; Kawai et al. 1993; Pellacini et al. 2002], for instance). The geometry-dependent lighting method by [Lee et al. 2006] may also be a useful indirect light design tool for visualizing scientific data. Okabe et al.[2006] and Akers et al.[2003] take other approaches to modifying lighting, providing an intuitive painting method for modifying the illumination of 3D models.

Our approach is inspired by all of the above methods. However, ours is unique in that it allows a user to add light and shade by painting them directly onto 3D objects without elaborate lighting control, to make stylistic animation by key-framing. In addition, we demonstrate that continuous tone detail can also be painted and animated as an extension of our approach.

3 Algorithm

3.1 Overall process

We begin by restricting ourselves to 3D cartoon animation, where each shaded area is assigned a uniform color by thresholded Lambertian shading ([Lake et al. 2000]). Starting from a 3D scene created using conventional lighting and key-framing techniques, we consider how to locally add light and shade onto surfaces. In particular, we describe how to use a paint-brush metaphor to design the shaded area at keyframes. The painting process at a given keyframe involves interactively adding light and shade details or sculpting the shapes of shade boundaries. Such editing is straightforward with our technique, while it would be very time-consuming and difficult to manage using conventional lighting.

Our implementation is capable of dealing with deforming geometry and multiple directional and/or point light sources; however, without loss of generality, we explain our idea below in the context of

a single light source. The extension to deformations and multiple light sources is straightforward. For a given threshold $0 < d_0 < 1$ a thresholded Lambertian shader creates two (possibly disconnected) regions, which we will call the *light* and *dark* areas. More precisely, using set notation we define the *light* area $\mathbf{B}_0$ on a surface $\mathbf{S}$, for a given threshold d_0 to be:

$$\mathbf{B}_0 := \{\mathbf{p} \in \mathbf{S} \mid \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) \geq d_0\}, \tag{1}$$

where $\mathbf{L}(\mathbf{p})$ and $\mathbf{N}(\mathbf{p})$ are the unit vectors representing the light direction and surface normal at a point $\mathbf{p}$ on $\mathbf{S}$, respectively. The boundary between light and dark areas is obtained by replacing inequality ($\geq d_0$) with equality ($= d_0$) above. We will refer to the dot product $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p})$ in (1) as the *intensity distribution*. Given these definitions, let us consider how to enlarge a portion of the light area, for example on the character's face in Figure 2, where the light area $\mathbf{B}_0$ is flesh colored. Let the area $\mathbf{C}_0$ with boundary $\partial\mathbf{C}_0$ (drawn in red in Figure 2) be an area painted with our brush-type interface (see the next section for specifics). The area $\mathbf{C}_0 - \mathbf{B}_0$ is the area that the user wishes to add to the original area $\mathbf{B}_0$. The core idea behind our approach is to *modify the intensity distribution* in order to make the light area change as desired, *i.e.* so that it becomes $\mathbf{B}_0 \cup \mathbf{C}_0$. The intensity distribution is a scalar function, so this greatly simplifies the problem when compared to working directly with light vectors and normals. The overall strategy is as follows. We first construct an *offset function* $o_1(\mathbf{p})$ defined globally on $\mathbf{S}$. This prescribes the new light area by replacing the original intensity distribution in (1) with $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_1(\mathbf{p})$(see Figure 2). Note that, though globally defined, the offset function should be mostly zero except in the region immediately surrounding the desired edit.

After making a modification at one keyframe, we can create a different offset function to define the light area at a second keyframe. By smoothly interpolating the offset functions between keyframes, we can achieve smooth animation of the light areas between frames as well. The procedure can be repeated for every pair of adjacent keyframes, resulting in an animated light area on $\mathbf{S}$ using just local edits with a paint-brush.

3.2 The offset function and key-framing

Next, we describe how to construct the offset function for a "painted" light area. Given the original light area $\mathbf{B}_0$ from (1) and the painted area $\mathbf{C}_0$, as shown in Figure 2. The offset function $o_1(\mathbf{p})$ for $\mathbf{B}_0 \cup \mathbf{C}_0$ should satisfy

$$\mathbf{B}_1 := \{\mathbf{p} \in \mathbf{S} \mid \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_1(\mathbf{p}) \geq d_0\} = \mathbf{B}_0 \cup \mathbf{C}_0, \tag{2}$$

where $o_1(\mathbf{p})$ is generated when the user finishes drawing $\mathbf{C}_0$. To fulfill condition (2), it is clear that the offset function should take values that are equal to $d_0 - \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p})(\geq 0)$ on the new boundary $\partial\mathbf{C}_0 - \mathbf{B}_0$. On the other hand, to make the offset function "active" only in the neighborhood of $\mathbf{C}_0$, we wish to have an area $\mathbf{D}_0$, which includes $\mathbf{C}_0$, that limits the extent of the domain where modifications to the lighting are applied (see Figure 2). In our current implementation, the distance between $\partial\mathbf{D}_0$ and $\partial\mathbf{C}_0$ is controlled by a slider in the user interface. The size of this region gives the user a way to limit the scope of modification (also see the detail in section 5). Therefore $o_1(\mathbf{p})$ should minimally satisfy the following conditions:

$$o_1(\mathbf{p}) = \begin{cases} 0 & \mathbf{p} \in (\mathbf{S} - \mathbf{D}_0) \cup (\partial\mathbf{B}_0 \cap (\mathbf{D}_0 - \mathbf{C}_0)) \\ d_0 - \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) & \mathbf{p} \in \partial\mathbf{C}_0 - \mathbf{B}_0 \end{cases} \tag{3}$$

If we choose for o_1 a continuous function satisfying the above conditions, then the resultant area $\mathbf{B}_1$ will have a continuous boundary. We can consider the new shaded area $\mathbf{B}_1$, to have a "generalized" intensity distribution given by $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_1(\mathbf{p})$, instead of

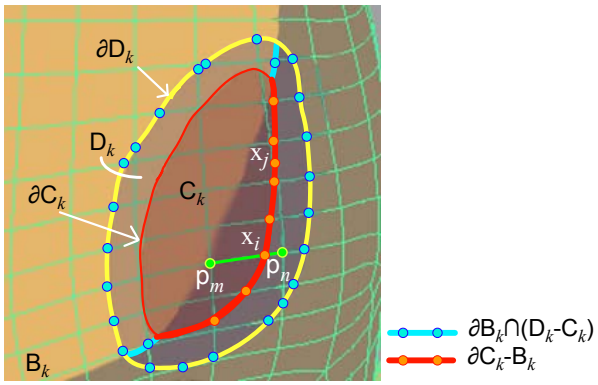


Figure 3: *The boundary constraint points used in finding the new offset function $\hat{o}_{k+1}(\mathbf{p})$. The orange points $\{\mathbf{x}_i\}$ take the value $d_0 - \mathbf{L} \cdot \mathbf{N}$, while the blue points are constrained to $o_k(\mathbf{p})$.*

$\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p})$. The above procedure can be repeated for each stroke, building upon the offset function created by the previous stroke. The user's kth stroke provides $\mathbf{C}_k$ and $\mathbf{D}_k$. From this new input, the resulting light area can be defined recursively as:

$$\mathbf{B}_{k+1} := \{\mathbf{p} \in \mathbf{S} \mid \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_{k+1}(\mathbf{p}) \geq d_0\} = \mathbf{B}_k \cup \mathbf{C}_k, \tag{4}$$

where we assume that $o_{k+1}(\mathbf{p})$ is a continuous function satisfying the constraints:

$$o_{k+1}(\mathbf{p}) = \begin{cases} o_k(\mathbf{p}) & \mathbf{p} \in (\mathbf{S} - \mathbf{D}_k) \cup (\partial\mathbf{B}_k \cap (\mathbf{D}_k - \mathbf{C}_k)) \\ d_0 - \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) & \mathbf{p} \in \partial\mathbf{C}_k - \mathbf{B}_k \end{cases} . \tag{5}$$

$\mathbf{D}_k$ includes $\mathbf{C}_k$ and serves the same role for $\mathbf{C}_k$ as $\mathbf{D}_0$ does for $\mathbf{C}_0$. The conditions in (3) can be seen to be a special case of (5) if we define $o_0 = 0$. Again we note that, outside of $\mathbf{D}_k$, no modifications will be made to the lighting (i.e., $o_{k+1}(\mathbf{p}) = o_k(\mathbf{p})$). In the $\mathbf{D}_k - \mathbf{C}_k$ region, no modification will be visible under the current lighting conditions, but some modification may be visible when either the lights or the model are moved. Having a $\mathbf{D}_k - \mathbf{C}_k$ band allows for smooth transition from modified $o_k(\mathbf{p})$ values to the original values.

To make the above strategy computationally tractable at interactive rates, we represent the offset function $o_k(\mathbf{p})$ with a sum of Radial Basis Functions (RBF), denoted by $\hat{o}_k(\mathbf{p})$. Thus in practice we use:

$$\hat{\mathbf{B}}_k := \{\mathbf{p} \in \mathbf{S} \mid \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + \hat{o}_k(\mathbf{p}) \geq d_0\} \tag{6}$$

in place of $\mathbf{B}_k$, and the boundary constraint (5) is only discretely enforced at a finite number of points. The RBF approximation $\hat{\mathbf{B}}_k$ is made from the shaded area obtained by the paint operation. Rigorously, the boundary of $\hat{\mathbf{B}}_k$ may not exactly match that of the original painted area. To allow fine adjustment, we provide two additional types of brushes: an *intensity brush* and a *smoothing brush*, which will be described in section 3.4.

Keyframing: Modifications made according to the above algorithm integrate smoothly with standard lighting equations, and for many animations a single offset function o_k may suffice. However, in order to create more elaborate modifications, it is possible to create several keyframes, with a unique offset function $o_{k,f}$ at each frame f, leading to more complex animation of light and shade. Lighting of the animation as a whole can then be accomplished by interpolating the offset functions $o_{k,f}$. In our prototype we have used simple linear blending for this purpose, though more complicated blending functions are possible and worth exploring.

3.3 RBF approximation of the offset function

Suppose that $\mathbf{S}$ consists of polygon meshes, as shown in Figure 3. We will assume for simplicity that $\hat{\mathbf{B}}_k = \mathbf{B}_k$. After obtaining $\hat{o}_k(\mathbf{p})$ and $\mathbf{B}_k$ in (6), we want to find $\hat{o}_{k+1}(\mathbf{p})$, which satisfies the boundary conditions (5) at a finite number of discrete points. We find a set of such points $\{\mathbf{x}_i\} \in \partial\mathbf{C}_k - \mathbf{B}_k$ by the following procedure. For each vertex $\mathbf{p}_m$ inside $\mathbf{C}_k$, we check adjacent edges for intersection with the boundary $\partial\mathbf{C}_k - \mathbf{B}_k$. For each intersecting edge, linear interpolation between $\mathbf{p}_m$ and the vertex at the other end, $\mathbf{p}_n$, is used to determine the approximate location of the boundary point $\mathbf{x}_i$. Note that we record stroke data per-vertex only and reconstruct the stroke linearly, thus no edge can cross the boundary more than once.

Now let $f \equiv \hat{o}_{k+1}$. We find a continuous f satisfying (5) for $\{\mathbf{x}_i\}$ in the following form [Duchon 1977; Wahba 1990; Turk and O'Brien 1999]:

$$f(\mathbf{x}) = \sum_{i=1}^{l} w_i \phi(\mathbf{x} - \mathbf{x}_i) + \mathscr{P}(\mathbf{x}), \tag{7}$$

where ϕ is a radial basis function, $\{w_i\}$ are weights, and $\mathscr{P}$ is a polynomial whose degree depends upon the choice of ϕ. In our case, l is the number of the boundary constraint points shown in Figure 3.

We employ $\phi(\mathbf{x}) = \|\mathbf{x}\|$ as the basis function after experimenting with various options. This corresponds to the solution of a generalized thin-plate spline problem on $\mathbb{R}^3$ [Duchon 1977; Wahba 1990], and the curvature minimizing properties of this basis function seem to be well suited to this task. Satisfying a discretized version of (5) reduces to solving a linear system of equations for the unknown weights $\{w_i\}$, and the four coefficients of the linear polynomial $\mathscr{P}$ on $\mathbb{R}^3$.

3.4 Additional brushes

The previous sections described how we enable users to add and edit *light* areas using a paint-brush metaphor. In a similar way we can add and edit *dark* areas. In that case the only difference is the selection of boundary points used in (5). Instead of using $\partial\mathbf{C}_k - \mathbf{B}_k$, we use the opposite half of $\partial\mathbf{C}_k$, that is, $\partial\mathbf{C}_k \cap \mathbf{B}_k$. The user simply switches the editing mode from *light* to *dark*. In both cases, the paint brush is used for roughly specifying the shading boundary. We call this type of brush a *boundary brush*.

The boundary brush works well to get a desired shape, but the intensity distribution may not change as smoothly as desired. This can be due to the radial basis function we select or due to too many conflicting constraints. For example, we have seen in our experiments that even a smooth radial basis function may result in a rapidly changing intensity distribution in the area where the distribution contours are very close to one another. This may cause the resulting keyframe animation to look unnatural. For this case, we have created a *smoothing brush*. By painting on the surface with the smoothing brush, the offset values are filtered, while preserving the original value of $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p})$. In our implementation, the offset values stored per vertex are updated using a simple weighted average of values at connected vertices for each stroke operation. In this way we achieve shading effects that fade in and out more gradually and have smoother boundaries (see Figure 4).

In some cases it is useful to be able simply to add or remove an isolated light or dark area. For these situations we provide a simpler alternative to the boundary brush, which we call the *intensity brush*. This brush simply adds to or subtracts from the offset function o_k. The amount added is determined by a magnitude parameter and the radius of the brush. The magnitude is the amount to add to o_k along the centerline of the stroke. We fade the added intensity smoothly to zero at the edges of the stroke using a "smooth-step" cubic polynomial falloff.

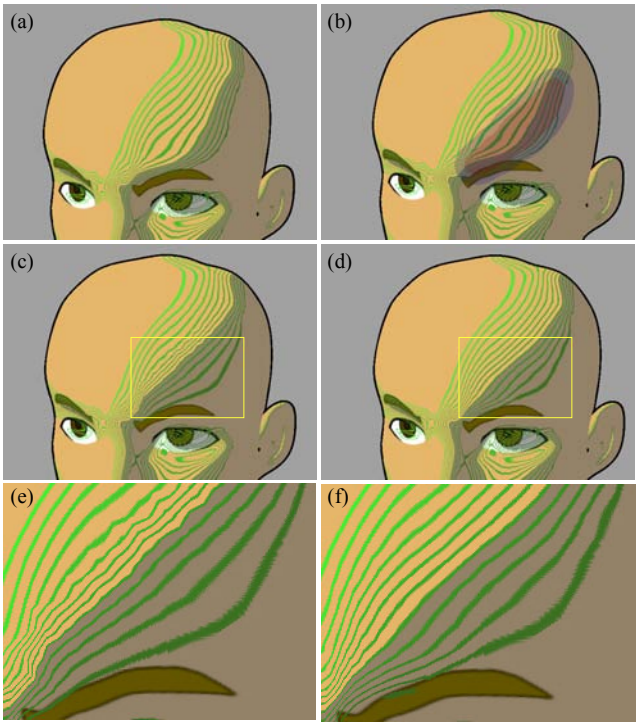


Figure 4: *Contours of the intensity distribution,* $\mathbf{L} \cdot \mathbf{N}$*, as influenced by our brush operations. (a) Initial distribution. (b) A boundary brush specifies a region which should become dark. (c) The new distribution with the offset function prescribed by the region. (d) The distribution modified by a smoothing brush. (e–f) Details from (c–d).*

Figure 4 shows a simple example of how to use these brushes. In Figure 4(a), an initial intensity distribution on the character is displayed using green contour lines. The boundary brush is then applied in (b). After getting the offset function in (6), we have the new intensity distribution as shown in (c). Using the smoothing brush, it is made smoother, as shown in (d).

3.5 Extensions

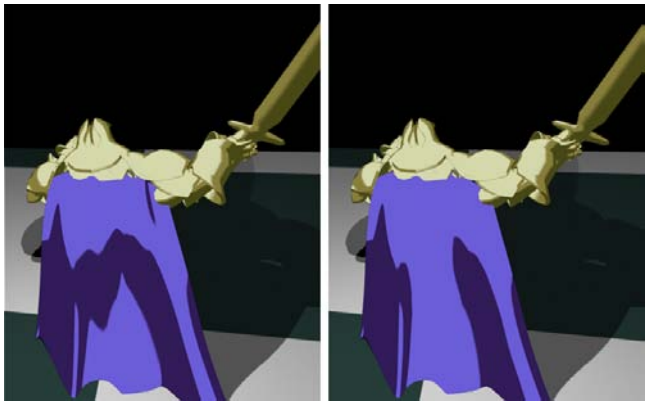
In order to get more variations of stylized light and shade, we add a few simple, but useful, extensions of the main algorithms above.

Specular Highlight: We can deal with stylized highlights in the same framework as the shaded area. In our system we simply need to replace the Lambertian term (the dot product, $\mathbf{L} \cdot \mathbf{N}$) in (6) with $\mathbf{H} \cdot \mathbf{N}$ from Blinn's specular highlight model [Blinn 1977], where $\mathbf{H}$ is the normalized half-way vector between the light and the eye. The user can easily edit the highlights by the brushes in the same manner as the shaded area.

Continuous tone control: The threshold d_0 in (1) is a global constant which controls the shaded area in accordance with (6), but this is not an essential assumption. Similarly, we can use the paint-brush metaphors to locally control and edit continuous tone on a surface by dispensing with the threshold and defining the lightness at a given point to be simply $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_k(\mathbf{p})$, or any continuous function thereof.

©YOUN IN-WAN, YANG KYUNG-IL/Shin Angyo Project 2004

Figure 5: *Editing shade and highlights. The animation* (left) *created using a standard cartoon shader was modified* (right) *using the techniques described in section 3. First the excessive highlight on the forehead was removed using the intensity brush, and then the boundary brush was used to create a light region around the chin, which was otherwise invisible.*



©2006 DELTORA QUEST PARTNERS

Figure 6: *Editing light and shade on a highly deforming object.* (left) *original frame.* (right) *edited frame. Using the intensity brush, we edited the light and/or dark areas on the deforming cape under rapidly changing lighting conditions. See also video 2.*

4 Implementation

Our prototype system is currently implemented as a Maya plug in, using Maya's hardware shader functionality that allows shader code to be written using standard OpenGL and GLSL. With our prototype system, the user can freely add localized light and shade to objects, and see the results, together with the conventional lighting, in real-time. In our GPU implementation, for each vertex i with position $\mathbf{v}_i$ on surface meshes, the offset function value $o_k(\mathbf{v}_i)$ is assigned and stored as a vertex color data in Maya, and is transferred from Maya to GPU as a varying parameter.

As for our paint-brush metaphor, we need to find all of the vertices inside the brush stroke region and calculate their distances from the stroke centerline. This information is used to determine the locations of the points on the boundary in Figure 3, as well as to implement the smooth falloff of the intensity brush. We accomplish this using a depth first search from seed points along the brush centerline. From each seed point, we find all the vertices with distance less than the brush radius, and set their distance values using the minimum of their current value and their distance from the current seed point. This data is needed only for the duration of a single stroke operation and can be discarded immediately afterward.

5 Results and discussion

We have applied our prototype system to making various stylistic animations. Our system currently runs at interactive rates on a 2.16GHz Intel P4 Core Duo CPU with an NVIDIA GeForce QuadroFX 350M GPU. In editing and previewing the animations, the frame rate ranges from 6 to 20 fps for all the examples in this paper and in the accompanying videos.

In making facial animation, controlling light and shade on the face is crucial. Figure 1 and the first half of the accompanying video 1 illustrate how effectively and efficiently our algorithms work for this important case. As shown in the video, even for making a simple facial animation, a 3D head model often creates many unnecessary dark areas, and it is very hard to remove them selectively using conventional lighting control. On the other hand, our approach can eliminate them easily and interactively. Moreover it allows the user to successfully add a variety of effects, each of which dramatically changes the character's impression.

Figure 5 and the latter half of the video 1 demonstrate a typical case where an animator uses our system to make the animation less realistic, but more expressive. Comparing with the animation under conventional lighting (left of Figure 5), we note several effects that have been added to the animation. Most obvious is the smoothing and simplification of the moving highlight on the protruding forehead. But also for example, the animator has added a light area to accentuate the jawline; a bright, firm line above the left eye; and delayed emergence of the face into the light, as shown in the right of Figure 5. Some of these effects might be achieved by conventional lighting techniques. However, it is almost impossible to add all of them into the same shot without resorting to frame-by-frame modifications.

Figure 6 and the first animation example in video 2 show the use of our techniques on an animated character with a highly deforming cape using a moving point light and a fixed directional light. This type of situation can result in light and shade areas that are distracting because they change too rapidly. The animation in the video demonstrates that our techniques are effective in eliminating such unnecessary shading and in simplifying light and shade to make it suitable for cartoon animation.

The second animation in video 2 demonstrates local controllability of continuous tone with our intensity brush described in section 3.5. As shown in the movie, even when adjusting the continuous tone on this object, our approach allows local tone control, adding a backlight effect around the character's shoulder (see Figure 7). We were able to create this animation without modifying the initial lighting setup. However, in cases where the viewpoint and/or lights are moving more dynamically, it may be more difficult to achieve the same effect using our technique.

In making these animations, we used either of boundary brush or the intensity brush, depending on the type of modification desired. The boundary brush is appropriate when the user wants to specify exactly where the new boundary should lie. If the goal is just to generally make a light or dark shape bigger or smaller, then the intensity brush is more effective. In the examples we determined the size of the paint brushes by experimentation. For example, we chose the width of the boundary brush so that one stroke of the

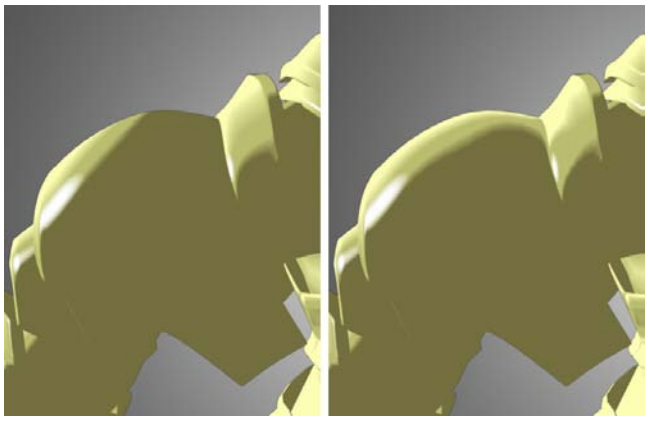


Figure 7: *Modifying shading with gradations. Here our prototype system has been used to make a directional lighting setup appear to be a more dramatic back-lit situation.*

#Verts	$\lvert\{w_i\}\rvert$	RBF(solve)	RBF(dist)	Transfer	Total
2011	68	0.63	5.0	38.8	44.4
8001	114	3.96	19.5	154	178.
31921	311	27.3	88	630	745.

Table 1: *Algorithm performance for strokes of various sizes. (All times in milliseconds). #Verts is the number of vertices in the stroke region. $\lvert\{w_i\}\rvert$ is the number of unknown weights in the RBF system being solved for, while RBF(solve) is the time taken to solve the linear system. RBF(dist) is the time taken to compute the RBF distance function for calculating $o_k(\mathbf{p})$. Transfer is the time taken to transfer vertex data to and from Maya in our plug in.*

brush includes at least two adjacent vertices of the surface mesh. Similarly, the distance between $\partial\mathbf{C}_0$ and $\partial\mathbf{D}_0$ in Figure 2, it is also set to include at least two adjacent vertices of the mesh, which can be accomplished using a slider. The small value of the offset function specified by the intensity brush in section 3.4 is also set empirically. Given the interactivity of our system, results of a particular parameter setting can be seen immediately, so we have not found it burdensome to search for these values via trial and error.

Table 1 shows the performance of our current implementation. The computation cost, however, depends on the number of vertices contained in $\mathbf{D}_k$. Since we do not paint very large regions $\mathbf{D}_k$ in practice, this cost seems not to be a serious bottleneck in our system. The most significant part was the basic cost of transferring vertex data between Maya and our plug in. The performance data in Table 1 also makes it clear that the algorithm itself is sufficiently fast for interactive editing.

Our prototype system has been made and tested in close collaboration with professional animators in our workplace since the very early stages of development. Initially, we gave a 20-minute tutorial to the animators. Since our system is implemented as a Maya plug-in, they were able to try it out on their own models immediately. The reaction has been positive - they do seem to find the system capable of producing the desired results easily and quickly. Most of the animations in the videos were designed with the animators so as to clearly display the capabilities of the proposed technique. Typically animations such as those shown in the videos take a few hours to complete, which is a drastic improvement over the previous techniques available to the animators. They also claimed that the conventional tricks such as texture animation or modifications to the character's geometry would make it difficult to maintain consistency between different shots with the same character. Therefore, with such conventional techniques, these kind of edits would simply be infeasible on a production schedule. Currently we are adding this system to an actual production pipeline, so it will soon be ready for use in forthcoming projects.

Even limiting the discussion to cartoon shading as shown in this paper, we still feel there are considerable applications of our algorithms not only in feature films, but also for television animation and even illustrative visualization. In contrast, the direct application of our method to interactive video games may be difficult; however, even in that context, it could be useful for non-interactive cut-scenes, since playback using our technique is lightweight and real-time on any modern GPU.

6 Limitations and future work

We have presented a few simple algorithms as steps toward a new methodology for truly directable stylistic depiction of light and shade in 3D animation. Our prototype system allows the user to locally and interactively edit light and shade by painting directly on 3D objects. Moreover the local edits integrate seamlessly with the conventional global lighting and animate smoothly regardless of the conventional lighting setup used. The animation examples and the videos illustrate these advantages over previous methods.

These algorithms, however, are exploratory. There are several things left to accomplish. In our approach, the RBF-based algorithm is used to obtain the rough boundary of the painted shaded area. In addition, we make the assumption that the vertices defining the object will not be added or removed during animation. We do not handle objects that change topology during an animation. We may need a more sophisticated algorithm to obtain a more precise approximation of the painted area. When applying this method to cartoon animation, highlights with very sharp edges are sometimes desired. But a smoothing RBF-based method cannot give such a sharp highlight directly. Providing boolean operations as in [Anjyo et al. 2006] may be of use here.

Our method allows us to add locally controllable light and shade, but at the same time conventional lighting control cannot be replaced by our approach. For example, as a very simple case, suppose that we want to move a small rounded highlight on a ball from one location to another. This could be easily accomplished by moving the light source. However, with the approach presented in this paper, the highlight would not move, but fade off at the original point, and fade in at the destination. This clearly demonstrates a difference between our approach and the conventional one. We believe that these approaches are complementary. Our approach is local, which means not only that it enables local editing, but also that the movement of light and shade is local.

We are currently investigating how to make cast shadows also locally controllable. We believe that a modified version of the approach described here has promise for achieving this. In this paper we have focused on the area of 3D stylized animation. However, this is an important practical area where there is a clear need for new techniques to help bridge the gap between artistic direction and the animator's heavy load. We hope our approach indicates a promising direction to serving such a practical need.

Acknowledgments

We would like to thank the SIGGRAPH reviewers for their substantial feedback to improve the paper. Many thanks also to Shinji Morohashi, Yosuke Katsura, and Ayumi Kimura for their dedicated help in making the animation examples. This work was supported in part by the Japan Science and Technology Agency, CREST project, and the first author was funded in part by grants from the Japanese Information-Technology Promotion Agency.

References

AKERS, D., LOSASSO, F., KLINGNER, J., AGRAWALA, M., RICK, J., AND HANRAHAN, P. 2003. Conveying shape and features with image-based relighting. In *IEEE Visualization. (Proceedings of Visualization2003)*, 349–354.

ANJYO, K., WEMLER, S., AND BAXTER, W. 2006. Tweakable light and shade for cartoon animation. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, 133–139.

BARLA, P., THOLLOT, J., AND MARKOSIAN, L. 2006. X-Toon: an extended toon shader. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, 127–132.

BLINN, J. 1977. Models of light reflection for computer synthesized pictures. *Computer Graphics 11*, 2, 192–198.

DUCHON, J. 1977. Splines minimizing rotation-invariant seminorms in sobolev spaces. In *Constructive Theory of Functions of Several Variables number 571 in Lecture Notes in Mathematics*, Springer-Verlag, 85–100.

GOOCH, B., AND GOOCH, A. 2001. *Non-Photorealistic Rendering*. AK Peters Ltd.

KALNINS, R., MARKOSIAN, L., MEIER, B., KOWALSKI, M., LEE, J., DAVIVN, P., M.WEBB, HUGHES, J., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: drawing strokes directly on 3D models. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH2002) 21*, 3, 755–762.

KAWAI, J. K., PAINTER, J. S., AND COHEN, M. F. 1993. Radioptimization: goal based rendering. In *Proceedings of SIGGRAPH 1993*, Computer Graphics Proceedings, Annual Conference Series, 147–154.

LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. 2000. Stylized rendering techniques for scalable real-time 3D animation. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, 13–20.

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

OKABE, M., ZENG, G., MATSUSHITA, Y., IGARASHI, T., QUAN, L., AND SHUM, H.-Y. 2006. Single-view relighting with normal map painting. In *Proceedings of Pacific Graphics 2006*, 27–34.

PELLACINI, F., TOLE, P., AND GREENBERG, D. P. 2002. A user interface for interactive cinematic shadow design. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH2002) 21*, 3, 563–566.

RUSINKIEWICZ, S., BURNS, M., AND DECARLO, D. 2006. Exaggerated shading for depicting shape and detail. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH2006) 25*, 3, 1199–1205.

SCHOENEMAN, C., DORSEY, J., SMITS, B., ARVO, J., AND GREENBURG, D. 1993. Painting with light. In *Proceedings of SIGGRAPH 1993*, Computer Graphics Proceedings, Annual Conference Series, 143–146.

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

TURK, G., AND O'BRIEN, J. F. 1999. Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH 1999*, Computer Graphics Proceedings, Annual Conference Series, 335–342.

WAHBA, G. 1990. *Spline Models for Observational Data*. SIAM.