Automatic Sprite Shading

Djalma Bandeira and Marcelo Walter
Centro de Informática - UFPE, Brazil
{dbs, marcelow}@cin.ufpe.br

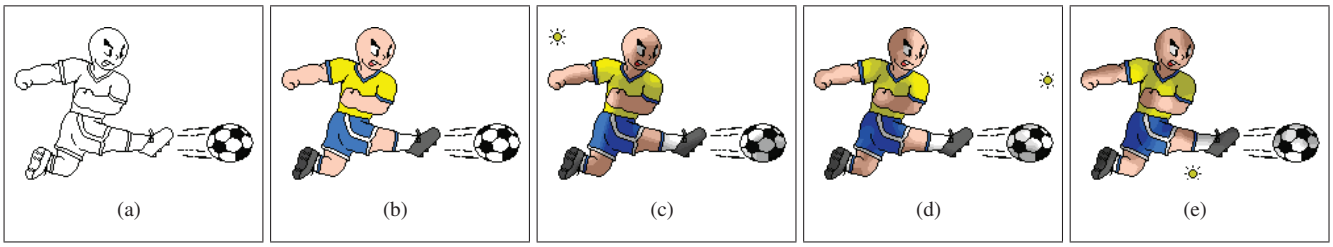

Figure 1: *From the outlines (a) we consider the basic colors (b) that are taken as input to generate different shading distributions (c, d, e) controlled by a local light source.*

Abstract

Sprites have been present since the first arcade games and console generations. Despite the advances in computer graphics and the whole representation of virtual worlds on 3D environments, 2D-based games still have their market, specially on portable consoles and mobile devices. The visual quality of today's games have increased as a result of hardware improvements in processing power, memory, and a richer color gamut. However, most of the editing task is still manual, using graphics editing tools. We present a method to automatically generate shading distribution on sprites, one important step during the editing process that is time demanding for most manual works on art design today. Our method allows to control the position of a local light source and to generate an approximation of the shading distribution effect. Although our solution is not physically accurate, according to the shape of the represented object, it can produce well usable results for 2D game systems in many practical cases, when compared with handmade sprite shading.

Keywords:: Real-time processing, art design, programming techniques, computer animation

1 Introduction

Two-dimensional games have a large impact on the entertainment industry today. Even with the evolution of 3D graphics, these games still assume an important role on the market. This is motivated for several reasons, from availability and low cost of mobile devices, to a requirement of simple but "catchy" graphics on games for kids. Besides, the development of native 2D games is generally less complicated than 3D games, specially on art related stages. The production is also widely explored by beginner game developers, for learning and application of basic 2D fundamentals.

Inside the game development process there is the art design stage, which is generally time consuming. For many projects, a team of game artists share the task of designing the visual identity for a game. In this stage, there are steps of graphics production and editing. This involves, among other things, art and effects on raster graphics, texture manipulation and, as the focus of our work, sprite editing.

The process of sprite generation and editing in game development is still a manual artistic composition for most cases. In fact, most of the literature describes the process as a time demanding procedure. It is generally presented as tutorials on websites and pixel art communities [Yu 2005; Tsugumo 2001; Sedgeman et al. 2004], although some basic concepts can be found on references such as [Feldman 2001]. Here we address the step of sprite editing related to the manipulation of shading effects, used to increase details on 2D images and make them look more like a 3D object, lit by a local light source.

Most of the solutions used nowadays rely on manual graphics editing on the image considering one or more fixed light sources. Alternatively, some solutions build a 3D coarse representation of the shape to better estimate and generate shading details, and then synthesize and express this information as a 2D image. Both methods have drawbacks when a new shading configuration is desired, requiring more image editing or another retrieval of shading information for the 3D representation in order to achieve the expected results.

We propose here an automatic solution to estimate and generate shading distributions on sprites which allows to control the position of a light source and experiment the results in real-time. Our approach, although straightforward, generates a visually consistent approximation of shading effects considering the light source. The results presented show good visual quality when applied on basic 2D sprites, which by nature do not require a highly precise shading distribution. Figure 1 shows the use of our method to automatically synthesize shading distributions controlled by the position of a local light source.

2 Related Work

To the best of our knowledge, there are no references involving shading manipulation directly related to sprites. However, more general approaches were presented, particularly focused on animations. One relevant work is [Johnston 2002] that presents a method to approximate and control the lighting distribution for 2D cel animation to aid the composition of cartoon live-action scenes. He uses a multi-channel image information to correct and obtain what would be a good approximation of normals, generating very convincing results for the final composed shading. This inspired the later work of Bezerra and colleagues [Bezerra et al. 2005], where they present a pipeline for 2D animation that also uses estimated normals from an outlined image to approximate and generate shading details, in addition with some simplifications from the previous work. According to the authors, the technique they proposed was the only method genuinely image-based so far and is easily applied to 2D cel animation.

Another work that deserves attention is [Anjyo et al. 2006]. They proposed a method to manipulate stylized light and shading animations in real-time, allowing the control of properties such as scaling, rotation, translation and splitting on shaded areas. Although flexible, their method still needs additional capabilities on tasks like making shaded areas as stylized highlights more editable. The work in [Todo et al. 2007] follows the same concept, presenting a set of algorithms to manipulate local shading distribution for a better consistence and enhancement on the final results. Although it is possible to obtain very good shading approximations with these methods, both approaches work based on 3D domain applications.

These related works present techniques on shading construction relying on 3D information or normal reconstruction to work, and while some may be suitable to be applied on sprites as well, they

generally demand several adjustments and/or layer constraints and manipulation for a good result. Our technique, on the other hand, provides a visual consistent shading approximation, exchanging the not so required physical accuracy on sprite images for a more flexible and fast method aiming both artists and beginners users. A good argument for our method is that what mainly differs a sprite shading from conventional shading effects on animations, is the fact that the former case generally considers an arbitrary light source not related to the external environment.

3 Method

The method we propose is divided into the following four stages, explained below:

1. Image Segmentation
2. Highlight Spots Approximation
3. Shading Distribution
4. Final Composition

3.1 Image Segmentation

We consider initially as input a sprite image containing only basic colors. In this first stage, we segment the image and separate the pixels that will be processed later. We first mark the background pixels with an uniform color as invalid pixels. The outline pixels themselves may or may not be also set as invalid pixels. This particular condition for outlines can alter the result drastically, as we use the key idea that the outline pixels separate distinct regions of the sprite and therefore determine basic shape and sections of the object being represented. This affects directly the shading distribution of separated small regions instead of considering the whole image as a single object without interior outlines. The final shading result will depend on the outlines distribution. On the other hand, ignoring the interior outlines can be useful in cases where a global shading distribution is desired.

The result of the image segmentation can be seen as a mask, where all the other colors are interpreted as valid pixels and represent the same region to be processed. Figure 2 below illustrates how the presence of outlines affects the mask and the final result:

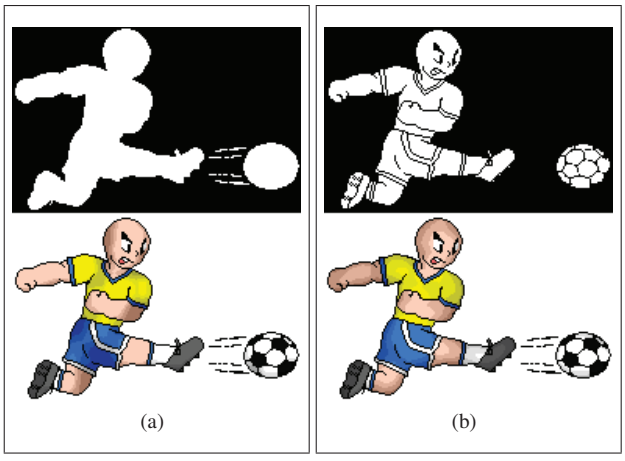

Figure 2: *Example of mask and the respective shading without (a) and with (b) outline constraint.*

3.2 Highlight Spots Approximation

Once the image is properly segmented, we now have to search for possible highlight spots. These spots are the regions of the image with the maximum exposure to the light source. First, we make image sections in one of the two axis directions (x or y) by tracking possible segments following the opposite axis. Considering initially the y-direction sectioning, we scan the entire image for x-segments as continuous lines of valid pixels. The segment ends when an invalid pixel is found or the bound of the sprite image has been reached. This may lead to several segments per line on the image. Since our method allows to control the position of the light source, we define the parameter l_p as the local position of the light inside the range $[0.0, 1.0]$, by taking the $[0.0, 0.0]$ position as the top-left corner of the coordinate system. Suppose that the end of a certain segment was found and p_0 and p_1 are the start and ending pixels positions, we calculate the center pixel p_c using a simple linear interpolation:

$$p_c = p_0 + l_p(p_1 - p_0)$$

Since l_p controls the light position, it consequently adjusts the center pixel on both directions to a position that "follows" the light source. We repeat this procedure for every single line of the sprite image and once all center pixels were computed, we have a sectioning on the y-direction. We restart this process by following now the x-direction looking for y-segments. Figure 3 shows each separated sections and the combination of both.

The next step is related to the shading weight of pixels. Now that we computed the sections, we classify every pixel in three levels of shading weight: weight 2 for those pixels inside the intersection of both X and Y segments, weight 1 for those pixels that belong to only one section and weight 0 for those out of any section. These weights determine the exposition of every pixel from the current light source, considering that those with weight 2 are the center of a highlight spot.

3.3 Shading Distribution

After the shading weight is properly assigned for every pixel, we now calculate the average shading distribution. We do this for all pixels by averaging the weight of every pixel inside the neighborhood of a certain regular window of size S_w. The window size is a parameter that varies according to the size of the sprite and the desired shading diffusion of a pixel among its neighbors. We consider setting the size $S_w = 7$ as a good value for examples around 100 pixels of resolution. Once the shading distribution in the whole image is processed, we blur it to reduce some sharpness that may affect the final result. We do this by averaging them again on a neighborhood window of size S_b that may be different in size from S_w. We set $S_b = 3$ for the case previously discussed, redefining both windows sizes to vary according to the sprite image dimensions. We can adjust both window sizes S_d and S_w independently, to better fit the resolution and desired shading effect. Figure 4 shows how we can improve the computed 3D look of two basic forms by adjusting the window size parameters.

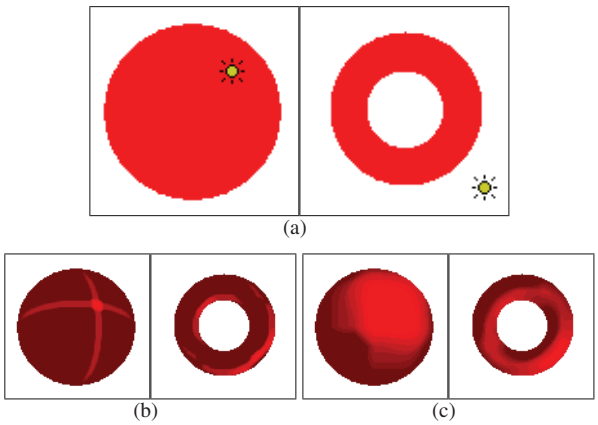

Figure 4: *3D shape approximation on shading of the input (a) can be improved from (b) to (c) by resetting the window values $S_w = 7$ and $S_b = 3$ accordingly. The parameters were reset to $S_w = 49$ and $S_b = 15$ for a sphere shape, and $S_w = 21$ and $S_b = 9$ for a torus shape. The yellow dot in (a) is the light source position.*

Once all average weights are computed, we reset all the values to the range $[0.0, 1.0]$ by considering the minimum and maximum average shading distribution found on the previous step to finally have

Figure 3: *Sectioning on the y-direction (a), x-direction (b) and the combination of both directions (c) with the highlight centers marked as red pixels. The green pixels are the center pixels for each segment.*

the estimated shading distribution. In Figure 5 we show some shading distributions considering the light source in different positions. Notice how the lighter and darker regions change according to the light position.

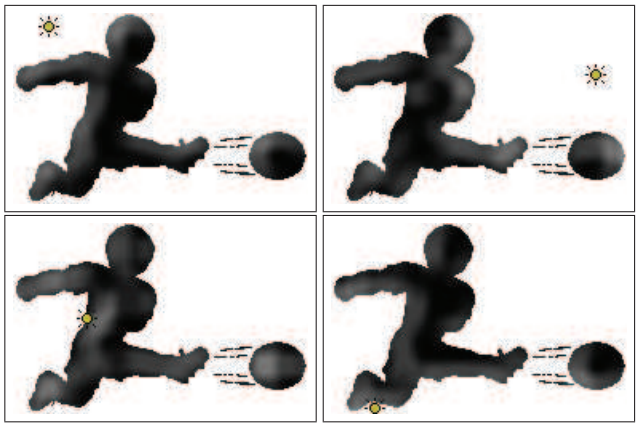

Figure 5: *Variety of shading distributions based on the local light source position (seen as a yellow dot).*

3.4 Final Composition

This last stage consists of combining the shading distribution previously computed to obtain different levels of shaded colors from the basic ones. We call these variations **color shades**, whose number per basic color is defined as the number of shades. To allow variable compositions, the color shades may be controlled by two parameters: **shading offset** and **shading shift**.

3.4.1 Shading Offset

The shading offset controls the distribution of color shades for every basic color inside the range $[0.0, 1.0]$ of the shading distribution. This parameter may be set evenly for every color shade considering the number of shades, or may be adjusted to change the balance distribution among them. Figure 6 illustrates different shading offsets for the basic skin color as well as the result on the entire sprite.

3.4.2 Shading Shift

The second parameter, shading shift (S_s), will control to which direction the variation of colors will assume for the current color shade. Let us consider a simple case of four shades for the previous sprite. The trivial case of $S_s = 0$ will treat the first color shade as the basic color of the sprite image among the possible colors. Lower values of shading distribution with this set leads to darker versions of the basic color. Now, if the shading shift is set to 1, as can be seen on Figure 7, the basic colors will be "shifted" to the right on the palette of possible colors. The same will occur for negative values, shifting to the left of the palette instead.

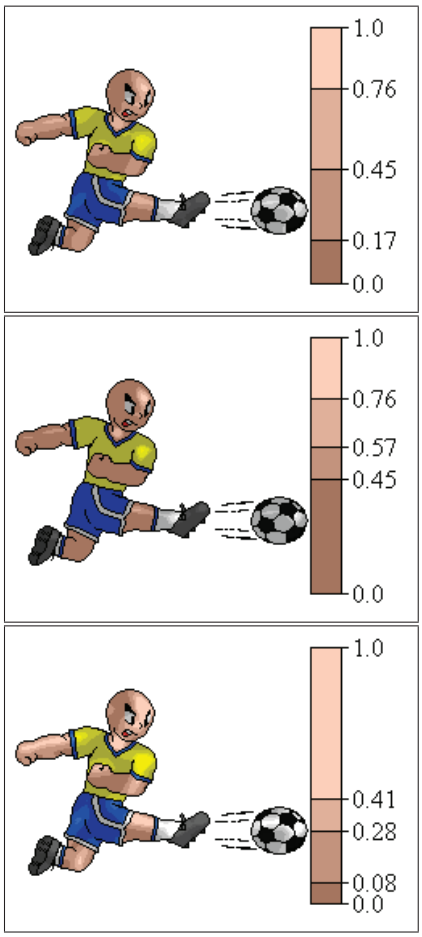

Figure 6: *Different shading distributions adjusted by the offset.*

3.4.3 Shading Composition

The two parameters discussed above will allow a wide range of shading variations. To calculate new colors from the basic color, we consider the shading shift S_s and a delta shading parameter ΔS. The delta shading is the percent of the basic color that separates different color shades. So, for every increase in the color shade, we have the corresponding color as a subtraction of the basic color.

Since we have the shading offset for each color shade, we can find from which shade S every pixel belongs to, by evaluating the value of its shading distribution. We then set a γ parameter as the value of the highest RGB component for the basic color. Now considering c as the color component and the previously discussed parameters, the final color $c\prime$ can be defined as:

$$c\prime = c - (\gamma \Delta S)(S - S_s)$$

Notice that the shading shift S_s adjusts the overall color variation by controlling positive or negative subtraction of the basic color

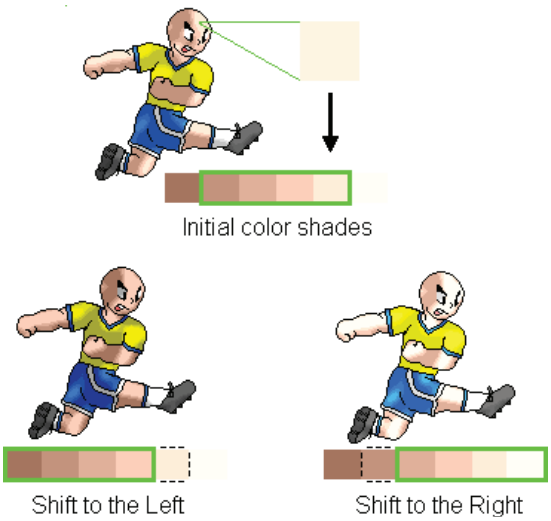

Figure 7: *One of the color palettes (skin color) with selected shades inside the green rectangle. The shift adjusts a new set of color shades on the palette.*

from the difference of the shade S. With the final color computed for every color component, the last operation is to clamp values outside the range $[0, 255]$ of possible colors on the RGB color space.

3.4.4 Extended Applications

After we have computed the shading information, other shading-based effects can be provided. One example is the highlight specular effect. Assuming that the shading distribution s of a certain pixel is higher than a threshold ω, we can obtain the new color c_h inside the specular region from the previous color $c\prime$ and the highlight level $\beta((s-\omega)/(1.0-\omega))$ as:

$$c_h = (1.0 - \beta)c\prime + \beta s_c$$

In this equation, β is the highlight level for the shading considering the distance from the threshold to the maximum value allowed 1.0. Then, we obtain the new color by balancing this highlight level between the previous color shade $c\prime$ and a specular color s_c. This is a simple approach we propose to control the effect with just two parameter tweaks, but other ideas can be elaborated and applied as well. Figure 8 shows and example using this method.

Figure 8: *Our application extended to handle specular highlights. The glass goblet sprite (a) without (b) and with (c) highlights. The parameter values used were $\omega = 0.4$ and $s_c = 0.9$ for all color components.*

Another extension for application using shading information is to generate dithering effects, instead of uniform color shades. We used the Floyd-Steinberg algorithm [Floyd and Steinberg 1977] to generate an alternative shading with less color shades, such as presented in Figure 9. The dithering effect as a shading representation with less color shades can be used for an alternative shading style or a palette simplification solution duo to hardware/software limitations.

Figure 9: *Sprite shading without (a) and with (b) dithering effect.*

4 Results

Most results presented in this paper were computed considering sprite images around 100 pixels of resolution. With minimum adjustments, we can handle specular highlights by considering that pixels whose shading distribution is over a certain threshold will assume a specular color. Figure 8 shows and example of this application on a glass goblet sprite.

Figure 10 shows a comparison between a manual shading effect and the automatic shading from our method. Notice that, even with the differences on shading distribution, our result is still visually compatible with a manual shading work and good enough for practical use. The computed result can later be refined by user manipulation. The manual artistic work may take a few hours whereas our solution is real-time, and besides, allows arbitrary light positions.

One possible drawback of an automatic solution is the shading coherence on sprite animation. Our approach, however, preserves this coherence, allowing batch processing without compromising the final work. Figure 11 shows a few frames of an animation loop after applying the automatic shading from the same light source.

Finally, in Figure 12 we show the shading processing with different parameter sets for shading shift and light source position to simulate alternative local light conditions.

5 Conclusion

We proposed an approach to estimate shading distributions on sprite images, considering adjustable parameters such as position of the light source and style related parameters involving variation of color shades. Although simple and fast, our method can be applied on a wide variety of sprites with a very flexible control of shading details at interactive rates. This ensures a better user experimentation and a faster achievement of desired results that on traditional techniques. The shading information generated allows the implementation of effects that extend beyond those presented here. Like [Bezerra et al. 2005], our method is genuinely 2D and the only one focused on sprite images. Since it does not require any 3D information, it is easy to implement and so highly recommended for beginner game developers to learn and test on their applications. Our technique is more recommended for low resolution sprite images, although its application on large sprites or even entire scenes may generate good shading results depending on the color distribution and/or outline details. Figure 13 shows how the processed shading can be evaluated as a normal distribution computed with a center light source and image gradient calculation from the shading. This is just to establish a comparison between the shading on the formal procedure and the use of recovered normals from the same shading.

We used the Phong illumination model [Phong 1975] for the normals with sphere mapping coordinates.

For future work, we will consider possible adaptations of the method for vector graphics, since games with vector-based images have drawn a lot of attention in the last years. We will also focus on improvements to approximate the shading coherence to the object shape.

Figure 10: *A basic color sprite (a) and the result of a manual shading work (b) and our method (c).*

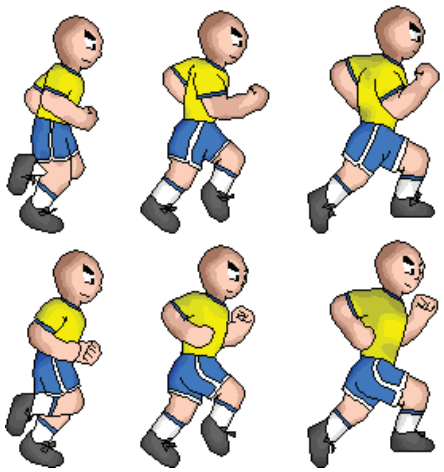

Figure 11: *Frame sequence of an animation loop. The shading coherence on the resulting animation is well preserved.*

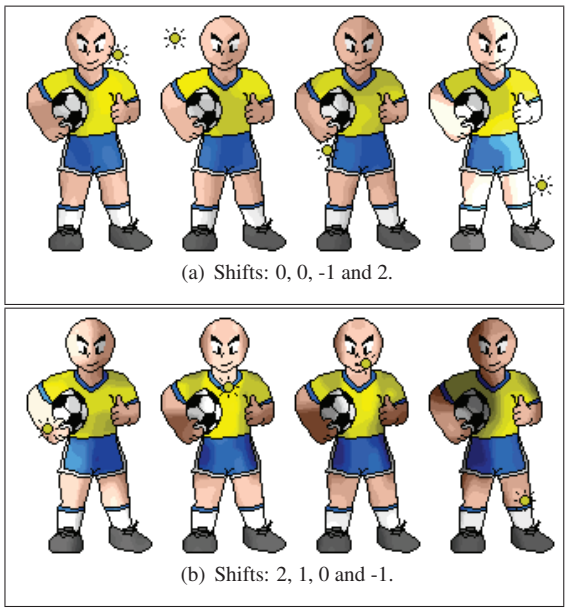

Figure 12: *Applications using 4 (a) and 8 (b) shades with variations on local light source position and shading shift.*

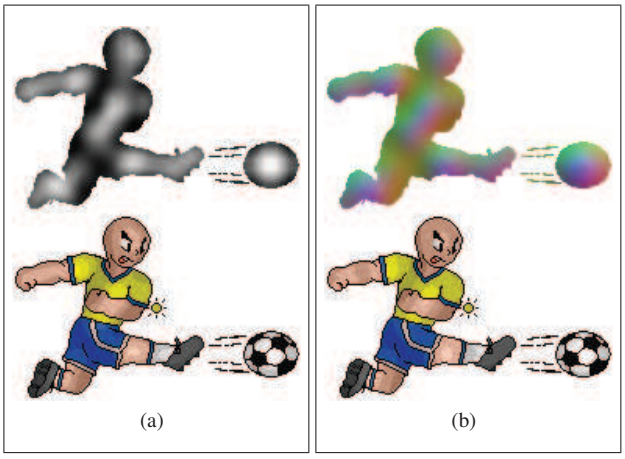

Figure 13: *An evaluation of normal approximation. The shading on a standard procedure (a) and using estimated normals (b) obtained from the same shading.*

Acknowledgements

We would like to thank Meantime Mobile Creations for helping us test the method by sending some state of the art game sprites (Figure 10). Work partially supported by FACEPE through grants IBPG-0216-1.03/08 and APQ-0203-1.03/06 and CNPq through grant 483356/2007.

References

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*, ACM, New York, NY, USA, 133–139.

BEZERRA, H., FEIJO, B., AND VELHO, L. 2005. An image-based shading pipeline for 2d animation. In *SIBGRAPI '05: Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing*, IEEE Computer Society, Washington, DC, USA, 307.

FELDMAN, A. 2001. *Designing Arcade Computer Game Graphics*. Wordware Publishing, Inc.

FLOYD, R., AND STEINBERG, L. 1977. An adaptive algorithm for spatial grey scale. In *Proceedings of the Society for Information Display*, 75–77.

JOHNSTON, S. F. 2002. Lumo: illumination for cel animation. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 45–ff.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Commun. ACM 18*, 6, 311–317.

SEDGEMAN, L., DAVIES, K., DIXON, I., VAN BRUGGEN, B., PLEIZIER, W., AND LEE, O., 2004. Pixel joint. `http://www.pixeljoint.com`.

TODO, H., ANJYO, K.-I., BAXTER, W., AND IGARASHI, T. 2007. Locally controllable stylized shading. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 17.

TSUGUMO, 2001. So you want to be a pixel artist? `http://www.petesqbsite.com/sections/tutorials/tuts/tsugumo/default.htm`.

YU, D., 2005. Derek Yu's Website. `http://www.derekyu.com`.