Sketching Shadows and Highlights to Position Lights

Pierre Poulin Karim Ratib Marco Jacques
Département I.R.O.
Université de Montréal
{ poulin,ratib,marcoj @iro.umontreal.ca }

Abstract

In inverse shading, a user provides information about the desired shading as he would like it to appear in the final image. The computer then interprets this information to identify the best values for the various shading parameters that would lead to the desired visual effect.

In this paper, we introduce an approach based on sketching in order to position light sources. Point light sources are positioned by sketches of shadows or highlights. Extended light sources are positioned by sketches of umbra or penumbra. The resulting system allows to quickly position light sources and to refine their positions interactively and more intuitively.

1. Introduction

Artists and designers have been *sketching* their ideas for centuries. In sketching, the general shapes and their relationships are first outlined, and then refined by adding smaller and smaller details. This is essentially an intuitive *top-down* process that generally leads to a faster design.

With the advent of solid modelers and computer-assisted design (CAD), a very different approach to designing has been introduced. The computer with its strict rules provides efficient and precise tools to design objects, but it does not easily lend itself to a process as intuitive and creative as sketching.

A few systems [11] [1] [2] have been presented which attempt to reintroduce the sketching concept as a part of the computer-assisted solid modeling process, but extracting the depth (third dimension) from sketches is often very difficult to handle correctly and intuitively. Sketches are 2D in nature and sketching 3D objects is quite difficult as depth with correct perspective must be accurate enough to be able to reconstruct the 3D objects from multiple views (2D sketches). With a single view, Zeleznik *et al.* [14] present an excellent sketching system based on gesture. Many of the ambiguous situations are removed by assuming most sketched edges are axis aligned. Their sketches are also used for shadows [4], but only to move objects rather than the lights themselves.

This paper studies sketching in modeling, but in a different context: inverse shading. The shading defines the way surfaces appear in an image. In our approach, points are interactively positioned, or sketched in a graphics 3D scene [3]. These points are considered to be in an umbra or a penumbra region, or in a highlight region. Then, by a system of constraints and optimizations, the *best* position for the light source is identified with respect to the sketched points. In a sense, the user indirectly manipulates light sources, hence the *inverse* shading aspect of the approach.

To position a light source, traditional systems specify light coordinates from input devices such as keyboard, sliders, or mouse. Even if shadows are displayed immediately, this approach requires the user to extract the light-blocker-shadow relationship to understand how a motion of the light will influence the shadow or the highlight. In some situations, small movements can drastically change a shadow, while in other situations, huge movements are necessary to slightly move a shadow. We believe letting the user interact directly with the scene and its visual effects (here we consider illumination effects) is a powerful concept. Directly manipulating shadows and highlights should provide an intuitive tool to quickly design the illumination in synthetic scenes, as long as the manipulation itself is intuitive. This is why we chose to use sketches of highlights and shadows. The system we propose should be at least as fast as the other systems in simple lighting situations and many times faster in more complex situations where the shadow or highlight position is very important. It will provide a solution at any scale for the light motion. The system will also inform the user when the desired shape of the shadow is impossible to achieve. The accompanying video illustrates the use and efficiency of our approach.

In the next section, we present an overview of inverse shading techniques. These techniques control the model (causes) of light interaction in 3D by a direct manipulation of their appearance (effects) in an image. Then, we describe the process of sketching points in a 3D scene. Once our context and goals will be established, we develop the necessary mathematics to demonstrate how sketches of shadow regions can be used to position a point light source. Then, sketches of umbra and penumbra regions are analyzed to position a predefined extended light source. Finally, we present a similar technique applied to sketches of highlights to position a light source and control the surface roughness.

2. Inverse Shading

The illumination model defines how much light reaches a given surface element and is reflected towards the view point to form an image of this surface element. While much research attention as been devoted to these illumination models, very little has been focused on helping a user to find the appropriate parameters values to lead to a specific appearance for a given scene.

Assuming a fixed scene geometry, Poulin and Fournier [10] paint color points onto surfaces. With a combination of constraints and optimizations, or weighted least-square fitting, their system provides values for the surface reflection parameters in the case of direct illumination. By restricting surfaces to be perfect lambertian reflectors, but considering all interreflections, Schoeneman *et al.* [12] change the lights colors by *spraying* color onto surfaces. In the same context, but allowing fewer surfaces to influence the final result, Kawai *et al.* [5] modify surface reflection and light emission, but based on more subjective impressions of the scene illumination. Although useful within their respective contexts, these techniques do not change the position of the light sources in order to obtain a shadow here or a highlight there.

Van Wijk *et al.* [13] define light direction by specifying the center of a highlight. Poulin and Fournier [9] extend this work in a modeler which controls light position and definition, as well as surface roughness, by manipulating directly within the modeler their effects (shadows and highlights) as if they were 3D objects. Their approach relies on the fact that these effects can be controlled by modifying interactively simple geometrical constraints, one at a time. However, while intuitive and effective within a modeler, these techniques still exhibit the general stiffness of the computer modeling process. For instance, transforming a shadow by a series of pivot operations can be at times cumbersome and may require many attempts to achieve the desired shadow shape. Positioning polygonal lights is also difficult due to the number of shadow volumes displayed simultaneously. We show in this paper how solving the same problem with sketching can greatly improve upon the design of simple illumination in a computer graphics scene. As in the earlier work of Poulin and Fournier [9], we concentrate our efforts on positioning lights, but this time from sketches of shadows and highlights. Highlights are also used to control the surface roughness coefficient.

3. The Sketching Process

In our context, sketches drawn on the 2D image plane project directly on the 3D scene as if they were pasted onto the surfaces [3]. The points are therefore in 3D, but display a simpler 2D nature because the perspective is handled directly by the renderer producing the image of the scene. Moreover in our modeler, we can identify all information about the structure of the closest object projecting in any given image pixel. Therefore, we can access information such as coordinates of points, surface normals, surface properties, light sources, etc.

We consider sketches as representations for shadows and for highlights. A sketched shadow of an object is composed of 3D points that are *all* enclosed by the real shadow cast by this object. Similarly, a sketched highlight corresponds to 3D points residing within the highlight. We consider also *anti-sketches*, where all anti-sketched points *must* all be outside the real shadow, or highlight.

The interface is rather simple. The user first selects an object and a light source. If no light is selected, a new one of desired type (point or polygonal) will be created. The user then sketches with the mouse points as continuous *strokes* into the 3D scene. After each stroke, a new position for the light is computed and its corresponding shadow is displayed. A sketch will attract a shadow while an anti-sketch will push it away. If there is no possible position for the light, the user is warned and he can delete the faulty stroke and add new ones.

For objects made of several primitives, the user selects a few basic elements and draws sketches for each of them (one sketch per element). The light position is then derived from these few elements rather than from the entire object. This makes the complexity of determining a position for the light manageable for any objects, while keeping the process simple and intuitive. Figure 1 shows a view of a simple scene where a sketch made of three strokes (in white) determines a position for a point light source (small white sphere above the ellipsoid). A solution was provided after each stroke, as illustrated by the images in the left to right order. The last image shows the result for another view point.

Now that we have a better feeling about the interaction in our system, we present next the underlying mathematics leading to these solutions.

4. Sketching Shadows from Point Lights

We first consider the process of positioning a point light source by sketching the shadow of a known primitive. For simplicity, assume for now that the primitive is a sphere. For each point forming the sketch of the shadow, the point is considered in shadow of the sphere *if and only if* the point

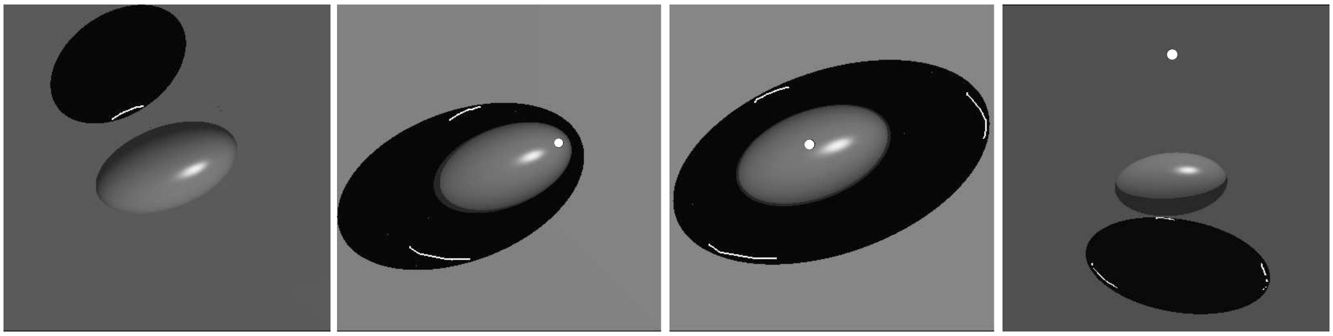

Figure 1. Sketches constraining the position of a point light source

light is located in the infinite portion of the cone defined by the sketched point and the silhouette of the sphere as seen from the sketched point. The sketched point (the summit or tip of the cone) and the point light must therefore reside on opposite sides of the sphere. A stroke of sketched points and a cone for a single sketched point are illustrated in figure 2.

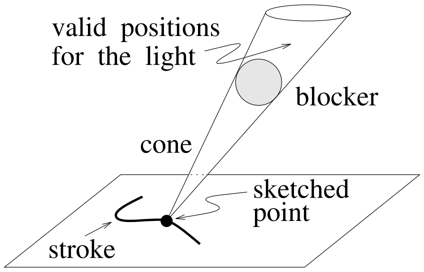

Figure 2. A stroke and a cone for one sketched point

This cone is represented in implicit form as a general quadric in arbitrary position expressed by the equation

$$\begin{aligned} Ax^2 + By^2 + Cz^2 + 2Dxy + 2Eyz + \\ 2Fxz + 2Gx + 2Hy + 2Jz + K &= 0. \end{aligned}$$

We can rewrite this equation in matrix form as

$$p\mathbf{Q}p^T = 0$$

expanded as

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} A & D & F & G \\ D & B & E & H \\ F & E & C & J \\ G & H & J & K \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Assume a canonical representation of a cone such that the summit of the cone is at the origin, its central axis of unit length is along the Z direction, and the circular base has a unit radius. The equation of this canonical cone is

$$x^2 + y^2 - z^2 = 0$$

which is equivalent in matrix form to

$$\mathbf{Q_c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

We can transform this canonical cone to the general one defined by a point and a circular silhouette with a transformation matrix $\mathbf{M_c}$. In a general situation, the sphere casting the shadow can itself be transformed by a series of affine transformations (scalings, translations, rotations, etc.) represented by $\mathbf{M_s}$. The resulting general cone equation corresponds to

$$\mathbf{Q}' = \mathbf{M_s}^{-1}\mathbf{M_c}^{-1}\mathbf{Q_c}(\mathbf{M_s}^{-1}\mathbf{M_c}^{-1})^T.$$

Let $\mathbf{M}^{-1} = \mathbf{M_s}^{-1}\mathbf{M_c}^{-1}$ be defined as a general graphics transformation

$$\mathbf{M}^{-1} = \begin{bmatrix} a & e & i & 0 \\ b & f & j & 0 \\ c & g & k & 0 \\ d & h & l & 1 \end{bmatrix}.$$

The general form of $\mathbf{Q}'$ then leads to the parameters

$$\begin{bmatrix} a & e & i & 0 \\ b & f & j & 0 \\ c & g & k & 0 \\ d & h & l & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the general conic form for each entry of $\mathbf{Q}$ are

$$\begin{array}{ll} A = a^2 + e^2 - i^2 & F = ac + eg - ik \\ B = b^2 + f^2 - j^2 & G = ad + eh - il \\ C = c^2 + g^2 - k^2 & H = bd + fh - jl \\ D = ab + ef - ij & J = cd + gh - kl \\ E = bc + fg - jk & K = d^2 + h^2 - l^2 \end{array}$$

A shadow cast by a transformed sphere from a point light source encloses every sketched points if the point light source is positioned within the volume defined by the intersection of all corresponding cones, all this on the *opposite* side of the sphere.[1] The resulting volume of intersections can therefore be infinite or closed. If it is infinite, a directional light is computed; otherwise, a point light source is positioned. An example is illustrated in figure 3. Robustly computing these intersections is a difficult task [7], but fortunately, by stating the problem as a constrained optimization problem, we become not interested in knowing the intersection volume itself, but rather simply in determining if a point belongs or not to all the volumes simultaneously.

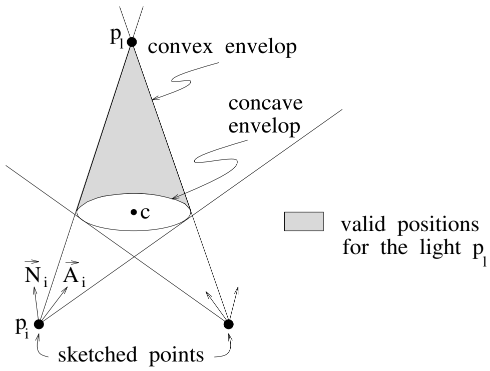

Figure 3. Convex/concave surfaces of an intersection volume

In order to identify a unique solution for the point light position, we need to identify a *consistent* optimization function that will produce intuitive results while still constrained within the intersection volume. We chose to maximize the distance between the sketched points and the light source. With this objective function, the shadow region bounds as tightly as possible the sketched points. The computed location for the point light corresponds then to the furthest point from the blocker. It resides on the convex envelop of the intersection volume.

While all of this could be computed analytically for a sphere, we opted for a general optimization algorithm. It requires however to provide as much as possible a valid initial position for the light. To do so, we compute the center of mass of the sketched points and associate the initial position of the light to an epsilon distance from the surface of the sphere along the direction from the center of mass to the center of the sphere. With this position, a non-linear (conics) constrained optimization [6] takes over to efficiently identify the furthest position for the point light within the volume of all cones. The implicit form of the conic equation is also useful because any point substituted in the equation produces a negative value if the point is inside the cone, positive outside and zero on the surface of the cone. The constraints, initial position and optimization function are given in table 1. Some of these variables are illustrated in figure 3. The optimization is completed on average in less than 0.003 second per sketched point on an SGI Indigo2 running at 150 MHz. Examples of the results are displayed for 1, 2, and 3 strokes in figure 1.

Therefore, a solution can be computed at interactive rates (less than one second) for about 300 sketched points, which is quite sufficient to control most shadows we experimented with. Because of redundancy, we observed that in our solver, the more points there are in a stroke, the less time it takes to perform the optimization per point. However if necessary, it is easy to reduce the number of sketched points under a convex blocker by keeping only the points residing *on* the 3D convex hull formed by all the sketched points of a stroke. This tends to reduce dramatically the number of sketched points per stroke, thus the number of cone equations and constraints. Finally in this situation, a solution always exists and is found, except when the sketched points lie on opposite sides of the blocker.

Extending the cones to other primitives is as easy or as difficult to derive as computing the silhouette equation for that primitive from any given point in space. For instance, with a convex quadrilateral, the cone is replaced by a truncated pyramid defined by the quadrilateral and four planes, each plane lying on the sketched point and one segment of the quadrilateral.

Care must be taken however if the object is concave or consists of several individual primitives. Rather than treating the entire object as an entity, we chose to treat each primitive individually (a sketch per selected primitive) and to compute the solution as the intersection of the solutions for the individual sketches. Thus the user can draw a sketch only for a single convex primitive of the object. However by selecting a few important primitives in the object, the computed light position will reside within all these intersection volumes. This allows to handle shadows of arbitrarily complex objects because only a few convex elements are used to control the entire shadow. For instance, to position a point light with the shadow of a tree, the user selects a leaf, sketches its shadow, and the solution is displayed. Then he selects another leaf to sketch its shadow. Each time, the light is more and more constrained, but at the cost of a few constraints rather than for all the primitives in the tree. For such objects, there are more situations for which no possible light position can be computed. However the incremental aspect of getting a shadow after each new stroke helps determining impossible ones.

[1] A point light within a sphere produces a shadow everywhere outside the sphere, but we consider this situation as undesirable.

constraints	$p_l \mathbf{Q_i} p_l^T < 0$	p_l is within cone i
	$\vec{N_i} \cdot (p_l - p_i) > 0$	p_l is above the plane $\vec{N_i}$ at p_i
$1 \le i \le n$	$\vec{A_i} \cdot (p_l - p_i) > 0$	p_l is on the right side of half-cone i oriented along the axis $\vec{A_i}$
initial position	$c + \left\lVert c - \frac{\sum_{i=1}^{n} p_i}{n} \right\rVert (r + \epsilon)$	just above an ellipsoid of center c and radius of length r along the alignment vector
optimization	$\max\left(\sum_{i=1}^{n} (p_l - p_i)^2\right)$	maximize distance from every sketched points

Table 1. Optimization parameters for a point light source p_l and n sketched points p_i

constraints for umbra	$\forall l, i \quad p_l \mathbf{Q_i} p_l^T < 0$
constraints for penumbra	$\forall i \ \exists p_l \quad \text{s.t.} \quad p_l \mathbf{Q_i} p_l^T < 0$

Table 2. Optimization parameters for an extended light source

5. Sketching Umbrae/Penumbrae from Extended lights

Objects casting shadows from extended light sources (linear, polygonal, polyhedral, spherical, etc.) produce three types of regions: completely illuminated, completely occluded (umbra), and partly occluded (penumbra). This latest region encloses the umbra by our definition.

Assume we want to position a predefined extended light by sketching the umbra/penumbra our transformed sphere casts from it. The sketched points of the umbra define, like in the previous section, a set of cones. However instead of forcing a single point light to reside within this volume, one must ensure *all* points of the extended light are enclosed by *all* the cones simultaneously. Optimization algorithms are used similarly by considering only the vertices of a convex polygonal light.

The sketched points of the penumbra define also a set of cones. However, instead of forcing all points of the extended light to be within all cones, the constraints are relaxed such that at least one point of the light must belong to each cone. Therefore if there is always a vertex of the light within each cone, the situation is identical as for the umbra. However testing for the vertices is not sufficient, as some portion of the light source could be within each cone while its vertices are all outside. We must then test, after trivially rejecting the vertices, that the intersection between the cone and the light is not null. This corresponds to substituting the light supporting plane equation into the cone equation, solving and making sure the 2D conic section is not included within the polygon defined by the light. This also can be generalized for extended non-polygonal lights.

Figure 4 shows the volumes constraining the position of an extended light for umbra and penumbra sketched points.

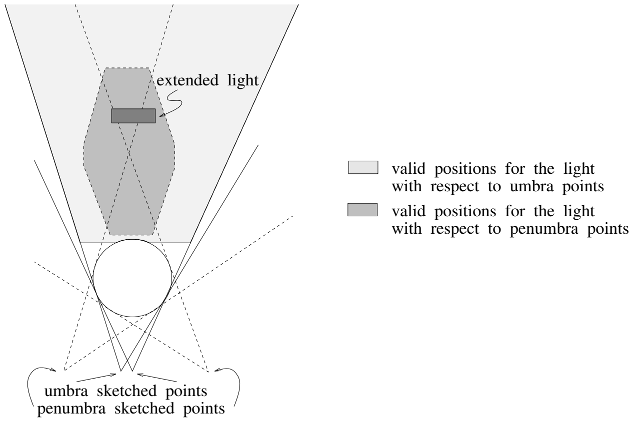

Figure 4. Valid positions for an extended light for umbra/penumbra regions

The constraints are of the form given in table 2. Figure 5 shows from two view points how strokes in white within an umbra (top) or a penumbra (bottom) constrained a rectangular light source positioned above the ellipsoid, represented as a white rectangle with black contour. The shadows are computed with shadow volumes from only light source vertices, which explains the discretized representations for the umbra and the penumbra.

The same techniques (non-linear constrained optimization) are used to find the position of the light that is the furthest from the sketched points. The performance results are also similar to those obtained to position a point light source from a sketched shadow, but degrades to about 0.01 second per sketched point when the light source is large with respect to the cones, due to more complex intersection tests between the extended light and each cone.

In our current implementation, we allow only for translations of the light source, although most rigid-body transformations could easily be considered. Transformations on the light shape are not allowed as this implies deforming the light source, which is beyond the scope of this paper.

Unfortunately with extended lights (unlike point lights), some configurations cannot produce any solution. For in-

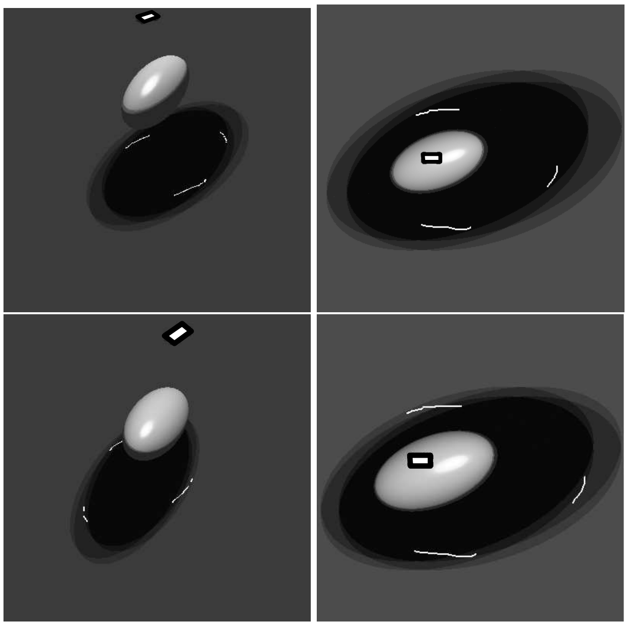

Figure 5. Sketches of umbra and penumbra for a rectangular light source

stance, a light larger than the blocker cannot produce an umbra region larger than the blocker itself. We found that this problem is usually avoided by starting with smaller penumbra strokes and refining the valid solution with more strokes.

6. Sketching Highlights

Another illumination effect corresponds to the presence of highlights onto specular surfaces. These highlights can also be used to control the position of a light source.

Highlights are usually simulated in reflection models by the specular term. In Phong's model [8], this corresponds to the function $(\vec{R_E} \cdot \vec{L})^n$, where $\vec{R_E}$ is the perfect reflection (mirror) of the viewer direction, $\vec{L}$ is the direction from the point to the light source, and n is the roughness coefficient of the surface. By our definition, a point on the surface is within a highlight region if the evaluation of Phong's specular function at this point is higher than a certain threshold t. This function is symmetrical around the direction $\vec{R_E}$ (as well as reciprocally around $\vec{L}$), and once cut at a threshold, it delimits a cone. The aperture of this cone is controlled by the roughness coefficient n and the value of the threshold t. Therefore, according to our definition, a point light source located within a cone produces a highlight at the sketched point if the point light resides inside every cones issued from every sketched points. Then the sketched highlight is completely enclosed by the real highlight.

Once again, the volume of the intersections can be infinite or can form a closed convex volume. By minimizing the value of the point light within the implicit cone equation,[2] we ensure that the highlight will be as tight as possible around the sketched points. The constraints on $\vec{R_E}$ and $\vec{N}$ at point p_i are identical to those respectively on $\vec{A}$ and $\vec{N}$ for the shadows (see figure 6).

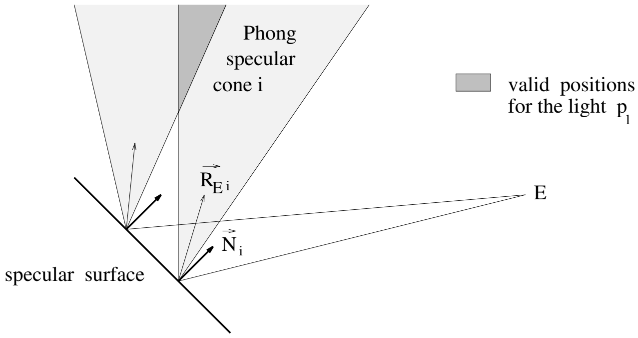

Figure 6. Visible region within Phong's specular cone

With all these constraints and an optimization function, the solver [6] can find a solution generally with the same efficiency as for shadows (under 0.015 second per sketched point). However for highlights, there are more frequent situations where due to the surface curvature and the cones aperture, two cones cannot intersect. By lowering the value of the roughness coefficient n, the aperture of each cone is increased leading to a higher possibility of a solution. The problem becomes then to find a value of n as large as possible so the sketched points are tightly bounded by the real highlight, but small enough so there exists at least one point residing in every cones. To achieve this within our optimization routine, we vary n from its original value down to 1, if necessary, using angular binary subdivision such that

$$n = \frac{\log t}{\log(\cos\theta)} \qquad \text{for} \quad \cos^{-1} t^{\frac{1}{n}} \leq \theta \leq \cos^{-1} t. \quad (1)$$

This is quite effective, but requires to iterate until $\Delta\theta$ is sufficiently small. Moreover, half of these iterations lead to no solution, for which the optimization routine spends more time computing various strategies to find a valid initial guess. The rate is still quite fast as stated above. Figure 7 shows two images with different sketched highlights and the corresponding rendered highlight, once the point light position and surface roughness have been computed.

[2]Remember that a negative value of the function indicates the point is inside the cone. Moreover, the value becomes smaller as it gets closer to the central axis of the cone and away from the summit.

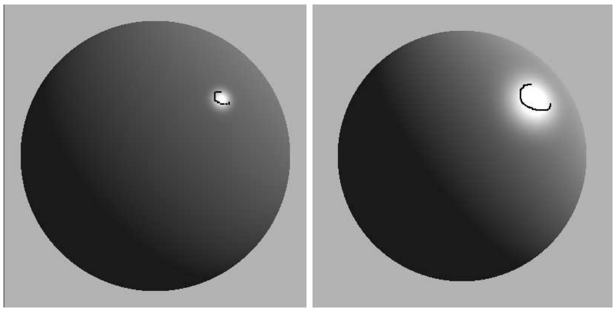

Figure 7. Two sketches and their highlights

7. Conclusion and Future Work

We presented a system to control the position of a light source by sketching its effects as shadows and highlights. The system is interactive and quite intuitive to use. It allows, by a series of constraints and optimizations, to adaptively control the solution by adding and deleting strokes. It can be used to either estimate a valid initial position for the light or to refine this position. Anti-sketch strokes with the opposite effects can also be used in a similar manner.

The final result is a system solving parts of the inverse shading problem. The impact of these inverse approaches will be directly related to the interface paradigm they are built upon. This is why we believe sketches in our context are well suited to the task of positioning the light sources. Moreover, these inverse systems can easily be used in conjunction with more traditional direct or menu-driven modeling tools.

We expect that with the major improvements of the graphics hardware in these recent years, more and more systems will use inverse techniques to let the user control the causes by manipulating the effects. As we are moving ever closer to the mixing of synthetic images with real images, better interfaces become crucial to efficiently drive traditionally computer vision techniques towards better image understanding. Interacting with the image as if it were the 3D scene, sketching to lead to the detection of important features such as shadows, highlights, textures, edges, etc. should provide faster and more accurate results, for the benefit of both vision and graphics communities.

Acknowledgements

This work originated from an interesting discussion with Dr. Toshimitsu Tanaka after a visit of the first author at NTT, Japan. We would like to thank Raphaël Mermond and Eric Larouche for contributions to the current implementation. We acknowledge financial support from NSERC, the Université de Montréal, Volvox and Taarna Studios.

References

[1] L. Eggli, B. Bruederlin, and G. Elber. Sketching as a solid modeling tool. In C. Hoffman and J. Rossignac, editors, *Solid Modeling '95*, pages 313–322, May 1995.

[2] I. J. Grimstead and R. R. Martin. Creating solid models from single 2D sketches. In C. Hoffman and J. Rossignac, editors, *Solid Modeling '95*, pages 323–338, May 1995.

[3] P. Hanrahan and P. E. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. In F. Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 215–223, Aug. 1990.

[4] K. Herndon, R. Zeleznik, D. Rubbings, D. Conner, S. Snibbe, and A. van Dam. Interactive shadows. In *UIST 92 Proceedings*, pages 1–6, Nov. 1992.

[5] J. K. Kawai, J. S. Painter, and M. F. Cohen. Radioptimization - goal based rendering. Proceedings of SIGGRAPH 93. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 147–154, 1993.

[6] C. Lawrence, J. L. Zhou, and A. L. Tits. User's guide for cfsqp version 2.3: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. Technical report TR-94-16rl, Electrical Engineering Department; University of Maryland, College Park, 1994.

[7] J. R. Miller and R. N. Goldman. Geometric algorithms for detecting and calculating all conic sections in the intersection of any two natural quadric surfaces. *Graphical Models and Image Processing*, 57(1):55–66, Jan. 1995.

[8] B.-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

[9] P. Poulin and A. Fournier. Lights from highlights and shadows. In D. Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 26, pages 31–38, Mar. 1992.

[10] P. Poulin and A. Fournier. Painting surface characteristics. In *Eurographics Rendering Workshop 1995*. Eurographics, pages 119–129, June 1995.

[11] D. Pugh. Designing solid objects using interactive sketch interpretation. In D. Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 26, pages 117–126, Mar. 1992.

[12] C. Schoeneman, J. Dorsey, B. Smits, J. Arvo, and D. Greenberg. Painting with light. Proceedings of SIGGRAPH 93. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 143–146, 1993.

[13] J. J. V. Wijk, F. W. Jansen, and W. F. Bronsvorrt. Some issues in designing user interfaces to 3D raster graphics. *Computer Graphics Forum*, 4(1):5–10, Jan. 1985.

[14] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. Sketch: An interface for sketching 3d scenes. Proceedings of SIGGRAPH 96. In *Computer Graphics* Proceedings, Annual Conference Series, pages 163–170, Aug. 1996.

Figure 8. Sketches to position a point light from its shadows; strokes in white are applied from the left image to the right image on their respective primitives