Crayon Lighting: Sketch-guided Illumination of Models

Amit Shesh
University of Minnesota at Twin Cities

Baoquan Chen
University of Minnesota at Twin Cities

Abstract

Appropriate lighting that highlights the salient features of a geometric model is critical to its visualization. In this paper, we propose and implement an interactive inverse lighting system in which the model is rendered based on sketched lighting effects. To specify target lighting, the user freely sketches highlights and dark regions on the model as if coloring it with crayons. Using the geometry of the model and these hints, the system efficiently derives light positions, directions and spot angles that attempt to achieve the hinted lighting conditions. As the system also minimizes changes from the previous specifications, lighting can be designed incrementally. The system solves the inverse lighting problem as an optimization problem and leverages commodity graphics hardware to make the process fast and interactive.

1 Introduction

Appropriate lighting of a model is critical to its visualization. Lighting enhances structural details to depict the geometry of a model well in a single image. However, given a model, its desired lighting is often achieved by moving lights around and "playing" with their colors and intensities in a largely trial-and-error fashion. This is a tricky process because traditional modeling systems that are meant for other high-level purposes assume lighting as a given, and so do not offer intuitive interfaces to set the lighting. Designing lighting in such systems is usually in the form of non-intuitive tweaking of actual coordinates and intensities. A method to automatically derive necessary lighting parameters based on a more intuitive specification of the desired lighting effects is desirable, and is referred to as the inverse lighting problem.

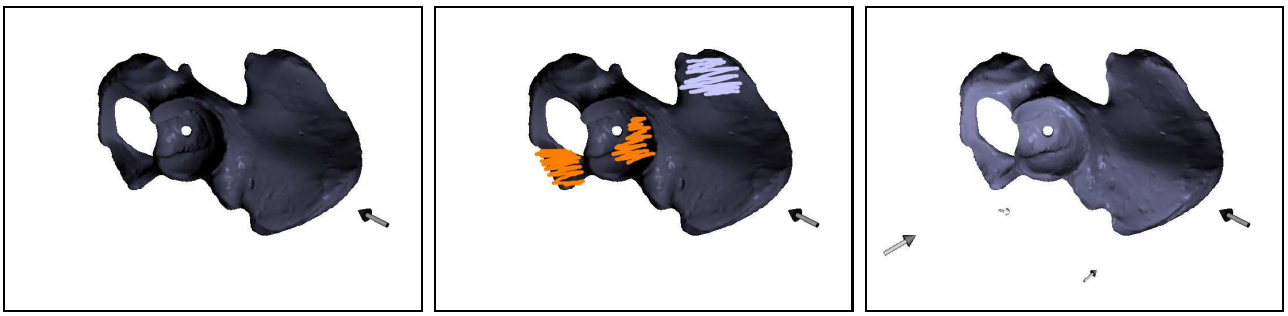
Many diverse approaches to solving the inverse lighting problem have been proposed and adopted. These approaches can be classified as being automatic or interactive. In the first category, a "good" lighting is procedurally inferred by using various perceptual metrics [3, 4]. There is little or no user intervention, and so speed is important but not critical. It is difficult in general to devise such metrics that work for all models and their geometric features, which is why most such approaches depend on several model-dependent parameters that can be set by the user. Other systems rely on interactive user input to determine a target lighting [5, 7]. These methods place the user in the driver's seat and conform lighting to the user's input. We contribute to this category of inverse lighting systems by proposing a simple sketch-based interface and using efficient data structures and hardware-assisted techniques to make the system fast.

As with rendering, global inverse illumination methods [3, 7] tend to be significantly slower but produce better pictures. However, in many applications, local illumination methods suffice to visualize a model–indeed many professional modeling tools support basic, point-light-based local illumination models. Many inverse illumination systems like those in [4, 5] use local illumination methods. Our work belongs to this category, and we mainly concentrate on point, directional and spot sources of light. Due to this, results from our work can be easily incorporated in simple graphics tools as well as larger systems that use point-light based illumination models.

The main motivation for our work is to develop a tool that can be used to quickly set up appropriate lighting for a model, given an illumination model. Our focus is to set up lighting quickly with intuitive user input. In this work, we first propose a sketch-based interface for general interactive inverse lighting systems. We process this input efficiently and use GPU-assisted techniques to solve the inverse lighting problem as a minimization problem. Our system handles point, directional and spot lights.

2 Overview

The user loads a model into the system with default lighting. The user uses an orange highlighting pen to sketch highlights on a part and a blue darkening pen for darkening parts of geometry by contrast (see Figure 1(b)). When the user input is complete, the system efficiently determines the affected parts of the model and the target lighting conditions. Various lighting parameters like positions, direc-



(a) (b) (c)

Figure 1. An example output of our system. (a) the original hip model with 40,000 triangles. (b) the user uses orange and blue strokes to bring the cavity into focus and recede the rear part by darkening it. (c) the result of the lighting (arrows show the light sources).

tions and spot angles of light sources are optimized to minimize the per-vertex differences between the actual and target lighting. When the optimized lighting is presented, the user can rotate the model, specify more constraints similarly and continue the design procedure. After satisfactory lighting is achieved, the system outputs all the relevant lighting parameters that can be plugged into any other program using a similar lighting model to reproduce the lighting. In this implementation, we adhere to the *OpenGL* lighting model, and thus the results produced by our system can be directly replicated in any rendering program that uses *OpenGL* lighting.

3 User Input

The user is equipped with two pen styles, a highlighter and a darkener (orange and blue strokes in Figure 1). The user sketches strokes to specify bright (highlights) and dark regions. Strokes can be in the form of hatching, cross-hatching or even solid coloring, as only the vertices they approximately cover are of importance. It is not necessary to stay within the silhouettes; the user can freely scribble on the model. Strokes can be retraced to emphasize greater brightness or darkness.

3.1 Determining affected vertices

We now efficiently determine the vertices marked by the user's strokes. Towards this purpose, we create a volume C_{ray} around the loaded model. Every voxel in C_{ray} contains a list of triangles that lie (at least partially) in it. In our current implementation, we use a 256^3 volume to achieve a reasonable trade-off between speed and memory requirements. In order to determine the vertices marked by the user, we cast rays from the eye through the marked pixels into C_{ray}. We use the fast voxel traversal algorithm proposed by Amanatides and Woo [1].

3.2 Determining the Target Light Field

The user's strokes are now interpreted to set up a target light field that the subsequent lighting must attempt to create. This light field is created by starting from the existing light field, and incrementing or decrementing vertex intensities according to strokes (if any) placed on them. Let S be the set of hit triangles obtained as explained in the previous section. Triangles in S by themselves do not represent the set of all affected triangles by the desired lighting parameters. We need a target light field that gradually changes over the model, starting from the hit triangles.

We pre-compute a score k_v for every vertex v in the mesh. We start with a default score of 0.1 for every vertex. For every edge e in the mesh, we calculate the gradient around it from the (at most 2) triangles that share it and increase the score of its end vertices accordingly. Thus, k_v is an indicator of the change in surface geometry around vertex v. We make increments and decrements of vertex intensities linear functions of k_v, thus enhancing each vertex according to its geometric context.

4 Solving the Inverse Lighting Problem

We now explain how various lighting parameters are obtained, given the target field generated as explained in the previous section. Various possible lighting parameters that can be solved for are light positions, intensities, directions and spot angles.

The general problem framework is similar to that suggested in [2, 3]: we formulate the problem as an n-dimensional non-linear optimization problem and solve it. Our minimizing function is of the form $f(X) = w_i * t_i$

where $w = [w_i]$ is a weight vector, and X is a candidate set of lighting parameters consisting of light intensities, directions, angles of spotlight, etc. Terms t_i measure the change in vertex intensities. For hit vertices, the intensities change according to the input, while for all other vertices, they remain unchanged. They are quantified by taking the maximum and sum of the difference between vertex intensities in the target lighting field and a candidate lighting field, respectively. We use Brent's minimization method [6] to solve this problem.

There are three primary issues with solving the above problem efficiently, which we briefly discuss in the following sub-sections.

4.1 Initial guess

As we solve for the position (in polar coordinates (θ, ϕ)), direction (dx, dy, dz), diffuse (k_d) and specular (k_s) intensities and spot angle ψ of a light, there are up to 8 parameters per each light source used. Solving such a multi-dimensional problem requires a good initial guess to minimize convergence to an incorrect local minima.

We surround the model with a sphere, tesselated into a fixed number of quadrilaterals (400 for results shown in this paper). Each quadrilateral serves as a bin of lights; there is at most one light source representing each bin. Every bin maintains a score that approximates the probability of a light being placed in that bin. When a vertex is sketched upon, we use its vertex normal to determine which bins are most and least probable to contain a light source that will produce the required lighting effect at that vertex, depending on whether the vertex is to be lit or darkened respectively. When all the input is processed in this way, we greedily choose the bins with the highest scores. The center of each such bin serves as an initial guess for the light contained in the bin.

4.2 Fast evaluation of $f(X)$

Every calculation of $f(X)$ requires calculation of the maximum and the sum of difference between two vertex intensities. This is an expensive operation because there are a large number of vertices. We delegate this calculation fully to the GPU to make it fast and efficient. This technique is inspired by the work by Windsheimer *et al* [8], who implement a visual difference metric in hardware.

Let V be a set of vertices over which f has to be calculated. We arrange vertices in V in a quad texture T of dimensions $\sqrt{|V|}x\sqrt{|V|}$. Every texel of this quad texture contains the difference between target and candidate intensities of a vertex. We pad the texture with 0's so that every dimension is a power of 2. We now read the texture locations $T(2x, 2y)$, $T(2x+1, 2y)$, $T(2x, 2y+1)$, $T(2x+1, 2y+1)$ and store their sum (or their maximum) as the color of an output texture $P(x, y)$. This texture is used as input for the next iteration. Finally, when the output texture contains only one texel, its value stores the required sum (or the maximum) that we simply read off. Thus this operation is completed in $O(\log_4 n)$ iterations, with $n = \sqrt{|V|}$.

5 Results and Conclusions

Figure 1 shows results of our system on the medical model of a hip with 40,000 triangles. Figure 1(a) shows the initial lighting conditions. The aim of the user input (Figure 1(b)) was to shift focus to the cavity in the model. The program correctly placed one directional source of light on the left of the model to create the highlight and others at the bottom to enhance the darkened region (Figure 1(c)). This result took 3 seconds.

Figure 2 shows results on some other models. In general several inputs could be required progressively to arrive at the desired result.

Figure 2(a) shows a pelvis model[1] with default lighting. The sacrum and coccyx area of the pelvis (marked by the rectangle) appears somewhat flat. The goal of the input in Figure 2(b) is to enhance the depiction of the geometry in this area. Figures 2(c-d) show the result using 4 sources of light. The grooves in the area of interest are enhanced because of the contrast produced by the lights.

Figure 2[2](e-l) show how lights can be obtained by progressively transforming a model and sketching on it. Figure 2(e) shows the filigree model having 177,000 triangles with default lighting. The goal is to contrast facets facing in opposite directions to arrive at a better lighting that enhances the structure of the model. The object is rotated and sketched upon in Figure 2(f) to obtain the lighting of Figures 2(g-h). Again, the object is rotated and some more hints are sketched (Figure 2(i)) to increase the contrast, as shown in Figures 2(j-k). Figure 2(l) shows how 8 light sources are placed (red arrow shows initial view point) to arrive at the final lighting in Figure 2(k).

In summary, "Crayon lighting" is a tool that performs inverse lighting given a sketchy input in which the user sketches highlights and shadows directly on the model. We envision this tool being used by modelers or researchers in computer graphics and visualization as a simple tool that outputs the necessary lighting parameters for good model visualization. Our system is easy to use and hence no prior learning is required.

[1] model obtained courtesy of VCG-ISTI by the AIM@SHAPE Shape repository

[2] model obtained courtesy of SensAble Technologies inc. by the AIM@SHAPE Shape repository

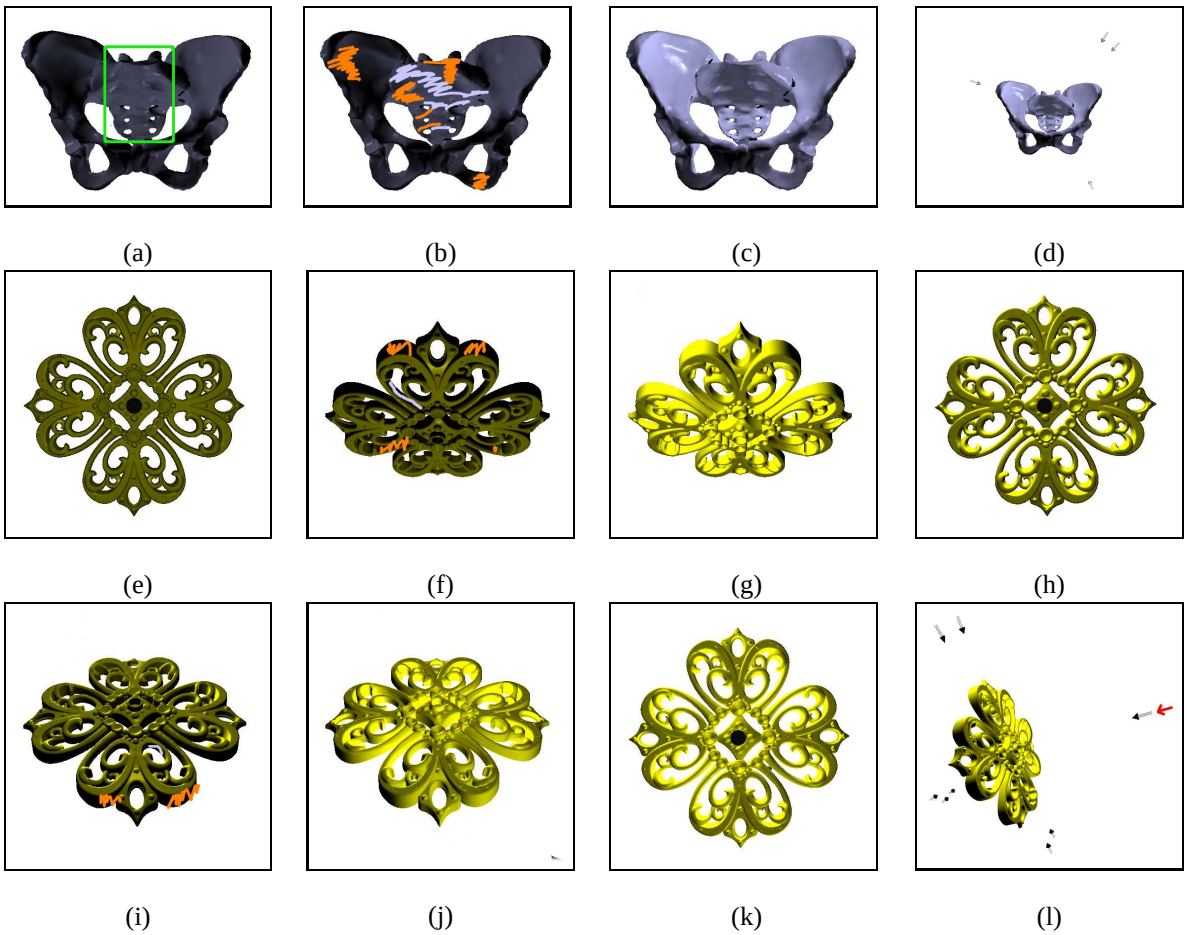


Figure 2. Results: filigree and pelvis. (a) a pelvis model with 50,000 triangles with default lighting. The sacrum and coccyx area marked by the rectangle appears flat. (b) the input specifying contrast in these regions to be increased. (c-d) the result showing the region enhanced by contrast. (e) a filigree model having 177,000 triangles with default lighting. (f) model is rotated and highlights and dark regions are specified. (g-h) result of input from (f). Some faces are better lit than in (e). (i) model is rotated again and more highlights and shadows are specified. (j-k) result of input from (i). Due to oblique and back lighting, the faces of the model are properly contrasted. (l) the same lighting in (k) shown from a different view point to see all the light sources. Red arrow shows the original view point in (e),(h) and (k).

6 Acknowledgements

Support for this work includes an NSF CAREER award (ACI-0238486). We thank Minh Nguyen for his technical inputs and Nathan Gossett for proofreading the paper..

References

[1] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Proc. Eurographics '87*, pages 3–10, 1987.

[2] A. C. Costa, A. A. Sousa, and F. N. Ferreira. Optimisation and lighting design. In *Proc. WSCG '99*, pages 29–36, 1999.

[3] J. K. Kawai, J. S. Painter, and M. F. Cohen. Radioptimization: goal based rendering. In *Proc. SIGGRAPH '93*, pages 147–154, 1993.

[4] C. H. Lee, X. Hao, and A. Varshney. Light collages: Lighting design for effective visualization. In *Proc. IEEE Visualization '04*, pages 281–288, 2004.

[5] P. Poulin and A. Fournier. Lights from highlights and shadows. In *Proc. SI3D '92*, pages 31–38, 1992.

[6] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, New York, NY, 2002.

[7] C. Schoeneman, J. Dorsey, B. Smits, J. Arvo, and D. Greenburg. Painting with light. In *Proc. SIGGRAPH '93*, pages 143–146, 1993.

[8] J. E. Windsheimer and G. W. Meyer. Implementation of a visual difference metric using commodity graphics hardware. In *Human Vision and Electronic Imaging IX, Proceedings of the SPIE*, volume 5292, pages 150–161, 2004.