Exploring Lighting Spaces

Tom Kang, Josh Seims, Stuart Shieber

TR95-18 December 1995

Abstract

We present a simple system for interactively specifying lighting parameters, including position, for high-quality image synthesis. Unlike inverse approaches to the lighting-design problem, we do not require the user to indicate a priori the desired illuminative characteristics of an image. In our approach the computer proposes, culls, and organizes a set of candidate lights automatically, using an elementary measure of image similarity as the basis for both culling and organization. The user then browses the set of candidate-light images, selects which lights to include, and combines them as desired. This work is a particular instance of a general strategy — sampling a design space broadly and intelligently and organizing the results for rapid browsing by the user — that may be applicable to many other design problems in computer graphics.

Exploring Lighting Spaces

T. Kang Harvard
J. Seims Harvard
J. Marks MERL
S. Shieber Harvard

TR-95-18 January 1996

Abstract

We present a simple system for interactively specifying lighting parameters, including position, for high-quality image synthesis. Unlike inverse approaches to the lighting-design problem, we do not require the user to indicate a priori the desired illuminative characteristics of an image. In our approach the computer proposes, culls, and organizes a set of candidate lights automatically, using an elementary measure of image similarity as the basis for both culling and organization. The user then browses the set of candidate-light images, selects which lights to include, and combines them as desired. This work is a particular instance of a general strategy — sampling a design space broadly and intelligently and organizing the results for rapid browsing by the user — that may be applicable to many other design problems in computer graphics.

1. First printing, TR95-18, January 1996.

1 Introduction

Lighting design for image synthesis is a significant and pervasive problem in computer graphics. Finding a good lighting design for a scene entails a search through the scene's *lighting space*, the space of its possible lightings. How this search should be conducted, and in particular what roles the computer and user should play in the process, is an open question.

The traditional approach to lighting design involves a tedious specify-render-evaluate loop in which the user is responsible for specifying all lighting parameters and for evaluating the results; the computer is responsible only for rendering images. This partition of work between user and computer is questionable, especially if sophisticated image rendering (e.g., ray tracing or radiosity) is part of the loop, making real-time searching of the lighting space impossible.

An alternative division of labor results from recasting the lighting-design task as an inverse problem: the user is now responsible for specifying objectives to be achieved in the lighting of a scene; the computer is responsible for searching the lighting space, i.e., for setting lighting parameters optimally with respect to the user-supplied objectives. For example, the user might specify the location of highlights and shadows in the image [3], or pixel intensities [4], or subjective impressions of illumination [2]; the computer determines lighting parameters that best meet the given objectives, using geometric [3] or optimization [2, 4] techniques. Unfortunately, the formulation of lighting design as an inverse problem has some shortcomings. Some are due to the inadequacy of current methods: for example, to make the computer's task tractable, the user may have to supply a limited set of fixed light positions [2, 4], thereby grossly limiting the region of the lighting space that will be searched. A more intrinsic difficulty is that of requiring the user to know and articulate a priori the desired illuminative characteristics of an image. This can require knowledge of lighting design, and skill in formulating suitable objectives.

The system we describe below allows large regions of the lighting space to be explored in a practical and intuitive way, and requires no special skill or knowledge to be used effectively. The roles that we assign the user and computer are different from those in the traditional or inverse approaches. In our approach the computer samples the lighting space by proposing, culling, and organizing a set of candidate lights automatically, taking into account hints about light types, placement, and directionality supplied by the user. This task is done as a batch process.[1] When it is complete, the user interactively selects and combines candidate lights (or rather the images they engender) to form a complete lighting design.

Our approach is described in detail in Section 2, and demonstrated on the accompanying videotape. The possible applicability of our general strategy to other design problems in computer graphics is discussed in Section 3.

[1]For the example we describe below, the batch processing, suitably distributed, would take overnight on five Sun SparcStation-10 workstations.

2 Technical Approach

The four elements of our technical approach are:

1. *Proposal:* Given a scene model and user-supplied hints about light types, placement, and directionality, a large number (typically several thousand) of different lights are proposed automatically. For each proposed light, a low-resolution thumbnail image is rendered using the desired rendering algorithm.[2]

2. *Culling:* Based on the thumbnail images engendered by the proposed lights, a set of candidate lights (typically numbering several hundred) is culled automatically from the proposed lights. The culling criterion is complementarity: the ideal set of candidate lights would be the one that best spans the space of possible illuminations. A higher-resolution image is then rendered for each candidate light.

3. *Organization:* The set of candidate lights is partitioned so that lights that produce similar illumination are grouped together. An index to this partition is generated to facilitate effective browsing by the user.

4. *User interface:* The user-interface allows the user to browse the set of candidate lights, to select individual lights from the candidate set, and to view linear combinations of the higher-resolution images engendered by those lights in real time.

The details of each element are described below.

2.1 Proposal

The input to our system is a scene model comprising viewing parameters, surfaces, and a set of light types. Some of the surfaces must be designated *light hooks*, and some must be designated *light targets*. The proposal algorithm works as follows:

- Generate M light positions distributed uniformly over the light-hook surfaces.

- At each light position, generate T lights of different types from the user-supplied set, such as point lights, area lights, and spotlights. Directional lights, such as spotlights, are aimed at randomly chosen points on light-target surfaces.[3]

- Render a low-resolution thumbnail image (e.g., 128×100 pixels) for each proposed light generated in the preceding steps, using approximately uniform radiant power for each light.

[2]We have used ray tracing as our rendering algorithm of choice. However, our approach is essentially independent of the rendering algorithm used.

[3]For the example described below, we took $M = 223$ and $T = 8$. The eight lights comprised one point light, one area light, and six identical spotlights. Of 15 large surfaces in the scene model, five — the walls and the ceiling — were designated light hooks and all 15 were designated light targets. This constitutes very little help from the user, whom we have supposed to be a novice. A lighting expert might be expected to give fewer and smaller light hooks and targets, and a larger set of light types. Our system readily accommodates differences in the quality and quantity of user-supplied hints.

Input:
P, a set of proposed lights and their images.
d, an average-luminance cutoff factor.

Output:
S, a set of n candidate lights and their images.

Procedure:

```
$CULL(P, d, n)$ {
    $P \leftarrow P \setminus \mathit{find\_dims}(d, P)$;
    $S \leftarrow \emptyset$;
    for $i \leftarrow 1$ to $n$ do {
        $\mathit{p\_score} \leftarrow -\infty$;
        foreach $q \in P$ do {
            $\mathit{q\_score} \leftarrow \infty$;
            foreach $r \in S$ do
                if $\mathit{image\_diff}(q, r) < \mathit{q\_score}$ then
                    $\mathit{q\_score} \leftarrow \mathit{image\_diff}(q, r)$;
            if $\mathit{q\_score} > \mathit{p\_score}$ then {
                $\mathit{p\_score} \leftarrow \mathit{q\_score}$;
                $p \leftarrow q$;
            }
        }
        $S \leftarrow S \cup \{p\}$;
        $P \leftarrow P \setminus \{p\}$;
    }
}
```

Notes:
find_dims(d, P) returns those lights in P with average luminance less than d.

Figure 1: The culling procedure.

Input:
S, a set of candidate lights and their images.
w, the hierarchical branching factor.
h, the desired number of levels in the hierarchy.
i, the current level in the hierarchy.

Output:
An indexed hierarchy of candidate lights and their images.

Procedure:

```
ORGANIZE(S, w, h, i) {
    $S' \leftarrow C \leftarrow CULL(S, 0, w)$;
    if $i \geq h$ then
        return;
    foreach $c \in C$ do {
        $k \leftarrow \sum_{j=1}^{h-i} w^j$;
        $S'' \leftarrow most\_like(k, c, S \setminus S')$;
        $ORGANIZE(S'', w, h, i+1)$;
        /* $S' \leftarrow S' \cup S''$; */
    }
}
```

Notes:
most_like(k, c, S) returns the k elements of S that are most like c, according to the image-difference metric incorporated in *image_diff*().

Figure 2: The organization procedure.

Thus the output of the proposal phase is a set P of $M \times T$ proposed lights and their corresponding thumbnail images.

2.2 Culling

The set of proposed lights will be too big and redundant for a user to browse it effectively, and the availability of only low-resolution images — there are too many proposed lights to generate higher-resolution images for all of them — would make it difficult for a user to evaluate individual lights, let alone combinations of lights. Culling the set of proposed lights on the basis of scene geometry is one possibility, but it appears fraught with difficulties. Instead, we perform culling on the basis of the lights' associated thumbnail images. The culling procedure is given in Figure 1. The first step is the elimination of lights that dimly illuminate the visible part of the scene, either because they are obscured or because they point away from the visible region: these lights are of little utility and can confound the rest of the culling process. Thumbnail images whose average luminance is less than a cutoff factor d are eliminated from the set P. Then a smaller set S of candidate lights is assembled by repeatedly adding to S the light in P whose thumbnail image is most different from its closest match in the nascent S.[4] The difference measure *image_diff*(q, r) is given by the expression:

$$\sum_{x,y} |Y_q(x,y) - Y_r(x,y)| \qquad ,$$

where $Y(x, y)$ is the luminance of the pixel at location (x, y). (We have experimented with several other measures based on the inner products of light images, but we judged them to be less effective in producing a useful set of candidate lights. Nevertheless, other difference measures, perhaps ones that reflect perceptual differences more accurately, may be more useful than our current simple measure.) Finally, for each light in the final set S, a higher-resolution image (e.g., 512×400 pixels) is computed.

2.3 Organization

We would like the size of the set S of candidate lights to be large, so that the user will have many complementary lights to choose from. However, the greater the size of S, the more difficult it will be for the user to browse the candidate lights effectively. The key to accommodating these contradictory requirements is to organize the set S into a suitable hierarchy. The recursive organization procedure is outlined in Figure 2. (For clarity of presentation, the control flow and parameter passing are described, but not the actual construction of the hierarchy data structure, which is straightforward.) At the top level, the $CULL()$ procedure from Section 2.2 is used to choose a set $C \subset S$ of complementary lights. Each of the lights in C is then used to "seed" a subtree that will contain the lights in $S \setminus C$ that are most similar to its seed. Two slightly different variants of the algorithm are possible: one in which duplicate lights are allowed in different subtrees, and one in which they are disallowed. The

[4]For the example below, we used a value of d that eliminated the dimmest third of the lights in P, and set $|S| = 584$.

difference between these variants is included in Figure 2 as a comment. This process is continued recursively until the hierarchy is populated completely.[5]

The parameters w and h (the branching factor and the number of levels in the hierarchy, respectively) are dictated by the user interface (see Section 2.4). In turn, these parameters determine the maximum number of candidate lights that can be accommodated: $|S| = \sum_{j=1}^{h} w^j$. However, when light duplication is allowed, the number of different lights in the hierarchy can be considerably less. This reduction in variety is usually offset by an improvement in the consistency of the hierarchic organization, which is important for effective browsing.[6]

2.4 User interface

The culling and organization of candidate lights allows the lighting space to be explored by rapid browsing. Our prototype interface, whose structure is depicted in Figure 3, presents to the user a row of eight thumbnail images that serve as the first level of the candidate-light hierarchy. Selecting one of these images causes its eight children in the hierarchy to be presented in the next row of thumbnails. The third and final level in the hierarchy is accessed in the same way.

Any of the images in the hierarchy can be copied using drag-and-drop into the palette at lower right. Higher-resolution copies of the thumbnails in the palette are combined (exploiting the additive nature of light [1]) to produce the final image, displayed at lower left. This step takes at most a few seconds, because the higher-resolution images were rendered previously during the culling step (Section 2.2). The contribution of any palette light to the final image can be controlled by a pop-up slider.[7]

A snapshot of the system in use is shown in Figure 4, and sample interaction sequences for the same subject scene are shown on the accompanying videotape. Note the differentiation of images within each row of thumbnails, and the similarity of the second- and third-level images to their ancestor(s) in the level(s) above. It is the coverage, consistency, and predictability of the automatically generated candidate-light hierarchy that makes efficient browsing of the lighting space possible.

3 Conclusions

The present technique for computer-assisted lighting design exemplifies a general principle for building computer-assisted design systems: Instead of solving very hard inverse problems

[5]The organization problem can be thought of as a form of graph partitioning, in which the images are vertices in a complete graph, and the edge weights are given by an image-difference metric. Graph partitioning is one of the most scrutinized problems in the field of combinatorial optimization, and more powerful (though more expensive) techniques than the kind of simple epitaxial-growth heuristic we use are known, and might be used in its stead to provide improved organization.

[6]In our prototype user interface, $w = 8$ and $h = 3$, allowing for 584 slots in the hierarchy. However, due to light duplication, which we currently favor, only 301 different lights are included.

[7]Our current user-interface design was developed with the novice user in mind. An experienced user would want to be able to manipulate the colors of the light sources, not just their intensities.

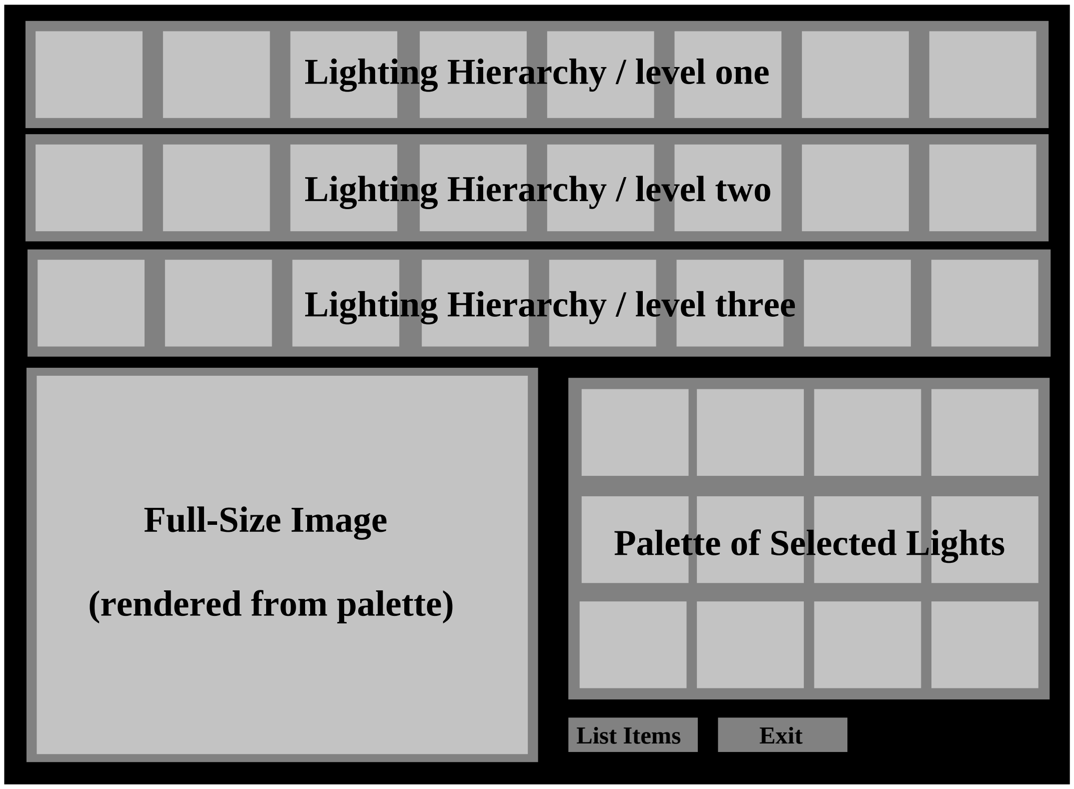
Lighting Hierarchy / level one
Lighting Hierarchy / level two
Lighting Hierarchy / level three
Full-Size Image
(rendered from palette)
Palette of Selected Lights
List Items
Exit

Figure 3: User-interface map.

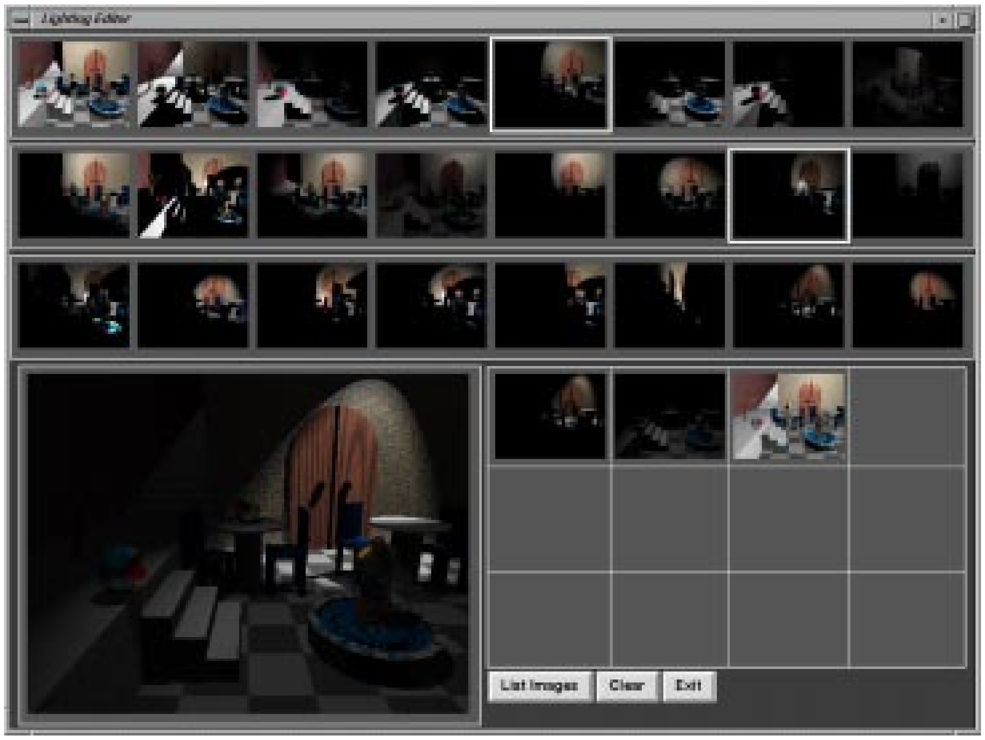

Figure 4: Screen dump of user interface.

directly, sample the corresponding search space broadly and intelligently and present the samples to the user in a well-organized, perceptually transparent manner. The sampling may require batch processing, but the user interaction will be in real time. When such a structured presentation is available, people can easily browse very large numbers of options, certainly in the hundreds and perhaps in the thousands or more.

Using this principle requires an ability to sample a search space effectively, and some measure of perceptual relatedness that can be applied to the samples. Fortunately, both are frequently available for design problems that arise in computer graphics. The technique could thus be applied to a wide variety of problems, such as: parameter selection for fractal, graftal, and other generative modeling processes; specification of the reflective and textural properties of surfaces, or the color and opacity attributes of volume data; motion synthesis for animation; and layout of 2D informational graphics.

4 Acknowledgments

Thanks to Bill Freeman, Sarah Gibson, and Kathy Ryall for commenting on early drafts.

References

[1] J. Dorsey, J. Arvo, and D. Greenberg. Interactive design of complex time-dependent lighting. *IEEE Computer Graphics and Applications*, pages 26–36, Mar. 1995.

[2] J. K. Kawai, J. S. Painter, and M. F. Cohen. Radioptimization – goal-based rendering. In *Proc. of SIGGRAPH 93*, pages 147–154, Anaheim, California, Aug. 1993. ACM SIGGRAPH, New York. In *Computer Graphics* Annual Conference Series, 1993.

[3] P. Poulin and A. Fournier. Lights from highlights and shadows. In *Proc. of the 1992 Symposium on Interactive Graphics*, pages 31–38, Boston, Massachusetts, Mar. 1992. ACM SIGGRAPH, New York. In *Computer Graphics* 25(2), 1992.

[4] C. Schoeneman, J. Dorsey, B. Smits, J. Arvo, and D. Greenberg. Painting with light. In *Proc. of SIGGRAPH 93*, pages 143–146, Anaheim, California, Aug. 1993. ACM SIGGRAPH, New York. In *Computer Graphics* Annual Conference Series, 1993.