Painting with Light

Chris Schoeneman
Julie Dorsey
Brian Smits
James Arvo
Donald Greenberg

Program of Computer Graphics
Cornell University
Ithaca, NY 14853

ABSTRACT

We present a new approach to lighting design for image synthesis. It is based on the *inverse problem* of determining light settings for an environment from a description of the desired solution. The method is useful for determining light intensities to achieve a desired effect in a computer simulation and can be used in conjunction with any rendering algorithm. Given a set of lights with fixed positions, we determine the light intensities and colors that most closely match the target image painted by the designer using a constrained least squares approach. We describe an interactive system that allows flexible input and display of the solution.

CR Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism; I.3.3 [**Computer Graphics**]: Picture/Image Generation; I.3.6 [**Computer Graphics**]: Methodology and Techniques - Interaction techniques.

Additional Key Words: simulation, global illumination, radiosity, ray tracing, lighting design, inverse problems.

1 INTRODUCTION

Although global illumination algorithms can produce strikingly realistic images, these algorithms can be difficult to use for lighting design. Currently the only tools available to designers are based upon *direct* methods—those that determine an image from a complete description of an environment and its lighting parameters. This forces a designer to begin with a geometric model, position the lights, assign their colors and intensity distributions, and finally compute a solution. The process is repeated until the solution matches the desired effect. This method is generally time-consuming, tedious, and often counter-intuitive. Given that we usually begin with a notion of the final appearance, a more natural, albeit more difficult, approach is to solve the *inverse problem*—that is, to allow the user to create a *target image* and have the algorithm work backwards to establish the lighting parameters. Inverse problems infer parameters of a system from observed or desired data [1]—in contrast with direct problems, which simulate the effects given all parameters. Although inverse problems are common in radiative transfer, thus far the field of computer graphics has been almost exclusively concerned with direct problems. Yet, inverse problems match a central goal of lighting design—determining how to achieve a desired effect.

In this paper, we present an approach that allows a designer to "paint" a scene as it is desired to appear. Given static geometry and a set of lights with fixed positions, a *constrained least squares* approach is used to determine the light intensities and colors that most closely match the target image painted by the designer. In the domain of lighting design, geometry often constrains the placement of the lights [2]; the designers frequently know about where to put the lights but not how the lights will combine or how bright to make them. Consequently, the task of selecting appropriate intensities for static lights is a useful subproblem of lighting design, and this is our focus. We do not address the automatic placement of lights, nor the mapping of simulated intensities to physical properties of the lights [3, 9].

2 INVERSE PROBLEM

The problem can be phrased more formally as follows: given static scene geometry and a desired appearance, determine the lights that will most closely match the target. There are constraints on possible solutions: only certain objects can emit light and only positive energy can be emitted—keeping us in the realm of physically meaningful solutions. The existence of constraints implies that not every target is realizable. The most general problem of determining how many lights to use, where the lights should be placed, as well as the distribution, color, and intensity of the lights is a non-linear optimization problem. However, if all possible lights have been positioned, and their distributions have been fixed, the determination of which lights to use and what their colors and intensities should be is a linear optimization problem.

2.1 Constrained Least Squares

Suppose $\{\Phi^1, \ldots, \Phi^n\}$ is the set of functions resulting from n distinct light sources illuminating an environment independently. These functions can be computed by any illumination algorithm, including those that account for interreflection and shadows. For example, they may be ray traced images [10] of a scene for each light from the same viewpoint, or radiance functions over surfaces in the environment computed via radiosity [4]. Let Ψ be the target function we wish to approximate. To formulate the approximation problem we require some minimal structure on the space of func-

tions. In particular, we require vector addition and scaling, which we define pointwise, as well as an inner product defined on pairs of functions (i.e. a symmetric positive definite bilinear form). From the inner product we gain the useful notion of the "size" of a function via the norm

$$||\Phi|| = \sqrt{\langle \Phi , \Phi \rangle}, \tag{1}$$

which provides a measure of error. The approximation problem can then be stated in terms of finding non-negative weights $w_1, \ldots, w_n$ such that the function

$$\widehat{\Psi} = \sum_{i=1}^{n} w_i \Phi^i \tag{2}$$

minimizes the objective function $||\Psi - \widehat{\Psi}||$. Stated in this way, the problem is one of least squares. Its unique solution is easily expressed in terms of the inner products:

$$\underbrace{\begin{bmatrix} \langle \Phi^1 , \Phi^1 \rangle & \cdots & \langle \Phi^1 , \Phi^n \rangle \\ \vdots & & \vdots \\ \langle \Phi^n , \Phi^1 \rangle & \cdots & \langle \Phi^n , \Phi^n \rangle \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} \langle \Phi^1 , \Psi \rangle \\ \vdots \\ \langle \Phi^n , \Psi \rangle \end{bmatrix}}_{b}. \tag{3}$$

The $n \times n$ matrix M is the Gram matrix of the inner product, which consists of the coefficients of the normal equations [7]. The Gram matrix is non-singular if and only if the functions $\{\Phi^1, \ldots, \Phi^n\}$ are linearly independent, which will normally be the case if all n lights produce distinct effects on the environment. Naturally, this excludes coincident light sources.

The remaining task is to define an appropriate inner product on the space of functions. Here we make use of the exact nature of the functions. If the functions assign intensities to a set of p discrete points, such as images consisting of p pixels, then the natural inner product is the p-dimensional vector dot product.

Alternatively, if the functions define surface radiance, the most natural inner product is the integral of the pointwise product of the functions. We further assume that the functions are piecewise linear, defined by interpolating a finite set of patch vertices. This representation is easily integrated yielding

$$\langle \Phi^i , \Phi^j \rangle = \sum_{k=1}^{v} \Phi_k^i \alpha_k^2 \Phi_k^j \tag{4}$$

where v is the number of patch vertices, α_k is proportional to the sum of all patch areas adjacent to the k^{th} vertex, and Φ_k^i is the radiosity at vertex k due to light i. Under these assumptions, the *normal equations* can be written

$$A^T D A w = A^T D \Psi \tag{5}$$

where A is the $v \times n$ matrix of the n vectors Φ^i, and D is the $v \times v$ diagonal matrix $\text{diag}(\alpha_1^2, \ldots, \alpha_v^2)$ of the weights used for the inner product. With this definition, $||\Phi||$ is proportional to the total power leaving all surfaces. Also, changes to the inner product are easily expressed as changes to D.

2.2 Solving the Normal Equations

The problem now is to solve the system of equations from Equation 5. This system contains n equations in n unknowns where n, the number of lights, is generally much smaller than the number of vertices in the environment or pixels in the image. Let $M = A^T D A$ and $b = A^T D \Psi$ as in Equation 3. We chose to solve the system $Mw = b$ using a modified Gauss-Seidel iteration.

There is no guarantee that the solution to the system has only positive entries. Simply clipping to zero after convergence is not a viable approach because negative values counteract some of the positive energy; ignoring them causes the environment to be too bright. To avoid this difficulty, we modify the Gauss-Seidel algorithm so that negative values are clipped to zero during each iteration. On the $k+1$ iteration of the modified algorithm, the updated value of w_i is

$$w_i^{(k+1)} = \max\left(\frac{b_i - \sum_{j=0}^{i-1} M_{ij} w_j^{(k+1)} - \sum_{j=i+1}^{n} M_{ij} w_j^{(k)}}{M_{ii}} , \ 0 \right). \tag{6}$$

Since a zero value does not influence other entries of w, we are effectively ignoring that light while the iteration is producing a negative value for it. In practice, this approach always converges in the sense that the difference between two iterations goes to zero. An alternative method may be found in [6].

3 IMPLEMENTATION

Our implementation is based on surface radiance functions as opposed to images. The system is therefore view-independent, solving for light intensities that are meaningful in a global sense, not simply for a given view. Although the system does no automatic placement of lights, the user may modify light source positions and distributions at any time. However, any such change requires that a new solution Φ_i be computed. To keep these operations fast, we have currently limited the solutions to direct illumination from each of the lights, accounting for distance and visibility but not secondary reflections. Similarly we restrict surfaces to be ideal diffuse reflectors. Using more complex techniques to find the light source functions makes moving a light more expensive, but does not affect the algorithm. By solving for the intensity of each color channel separately, the colors are determined as well as the intensities.

The user modifies the radiance function of the target by "painting" light onto surfaces. We also adjust the matrix D so that painted surfaces have more weight (or more area) in the solution, causing the system to try harder to match painted surfaces than unpainted ones. This is necessary in complex environments where the large unpainted areas can overwhelm the effect of small painted areas.

To achieve interactive speeds while painting we use the method introduced by Hanrahan and Haeberli [5] to quickly find which patch the brush is currently affecting. Object id's and the patch uv coordinates are rendered into auxiliary buffers. A lookup at the paint brush position in these buffers quickly identifies the patch being painted. Only painted patches are redrawn. Since very few patches change at once, updates are easily made in real time.

The patch's reflectance function modifies the light as it gets painted on a surface. This prevents a surface from being painted with physically unattainable colors. For example, a purely red surface cannot be painted blue. The modified light then gets distributed to the patch's vertices according to their proximity to the paint brush. We restrict the radiosity at a vertex to between zero and one and linearly map this to the full dynamic range of the display.

The system recomputes the closest fitting combination of lights after each brush stroke. All vertices painted between a button press and release comprise a stroke. To maintain interactivity, we perform all the updates incrementally. Instead of completely rebuilding Ψ

(the target radiosities) and re-solving, however, we only change the elements corresponding to painted vertices and make incremental changes to the inner products. If $\Delta\Psi$ is a vector of the changes to the radiosities with p non-zero terms, then

$$b_{new} = A^{\mathrm{T}}D(\Psi + \Delta\Psi) = b_{old} + A^{\mathrm{T}}D\Delta\Psi. \quad (7)$$

Since $\Delta\Psi$ is typically very sparse, we can update b with $O(np)$ operations by ignoring all zero entries of $\Delta\Psi$. Since most of the environment hasn't changed, the old intensities provide a good initial guess for the modified Gauss-Seidel iteration and it converges quickly. We can similarly update the weight (i.e. effective area) of vertices. Consider changing the importance of one vertex. Let ΔD be the diagonal matrix with its sole non-zero entry being the change in weight of the vertex. Then

$$b_{new} = A^{\mathrm{T}}(D + \Delta D)\Psi = b_{old} + A^{\mathrm{T}}\Delta D\Psi. \quad (8)$$

Because ΔD has only one non-zero entry, $\Delta D\Psi$ has only one non-zero entry and b_{old} can be updated with $O(n)$ operations. Changing the inner product, though, requires that M be updated as well. This can be done incrementally, observing that

$$M_{new} = A^{\mathrm{T}}(D + \Delta D)A = M_{old} + A^{\mathrm{T}}\Delta DA. \quad (9)$$

Since ΔDA has only one non-zero row, we need to look at only one column of A^{T} so we can do the multiplication in $O(n^2)$ steps.

In addition to painting, the user can also interactively move and aim light sources. Changing a light requires recomputing the direct illumination due to that light. Since A changes, M must be recomputed as well; however the cost of recomputing a column of A greatly overshadows the matrix multiply used to determine M. Because this can take time for large environments, the user can defer these computations until all the lights have been satisfactorily placed.

The user may also move the camera interactively. Because we paint directly onto the geometry, painted surfaces are view-independent. Also, since no directional effects are accounted for, the functions Φ^i for each light are independent of the position and orientation of the camera. Therefore we need not recompute $A = |\Phi^1 \cdots \Phi^n|$ or re-solve for the light intensities as a result of moving the camera.

4 RESULTS

We tested the system on a moderately complex environment consisting of polygonal meshes with about 19,000 polygons, 27,000 vertices, and 12 lights. Figure 1 shows the user's painted environment at the top and the system's solution on the bottom. A user can see both views at once while working to get immediate feedback on how closely the design is being met. Figure 2 shows the same environment with the same light positions but with different painted intensities and colors (left) and a distinct best approximation (middle). The lighting parameters determined by the interactive lighting design were then used to compute a ray traced solution, which is shown in Figure 2 (right). The large scale washes of color and illumination levels are captured well in the rendered image. The user can quickly and easily modify a design to have a very different appearance.

Figure 3 shows the screen during a painting session. The window in which the user paints is on the left and the best fit solution is on the right. Some of the support tools for choosing light to paint and positioning lights are also shown. In this design, 14 lights were placed in another environment of similar complexity.

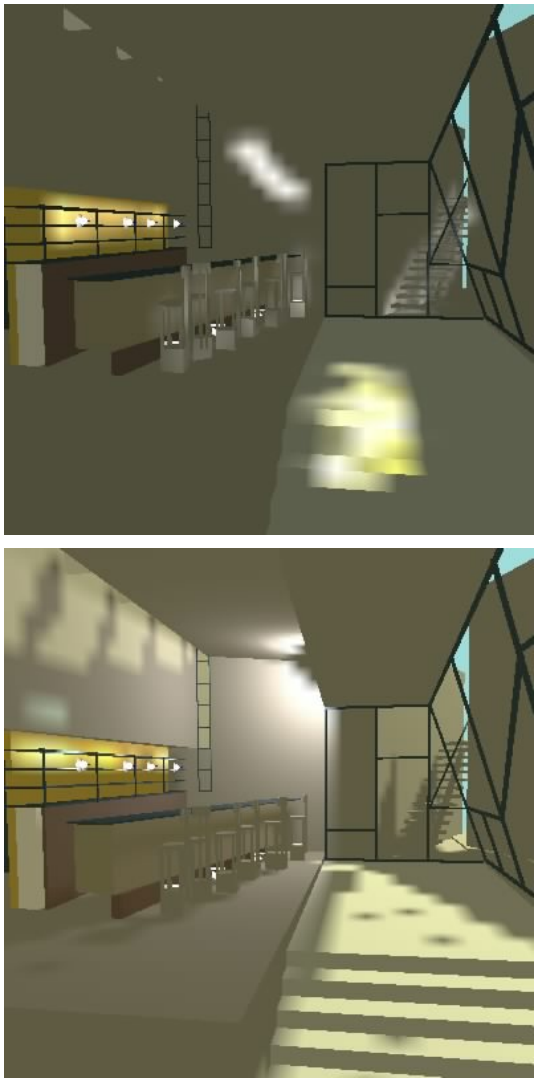

Figure 1: Design (top) and associated best approximation (bottom).

5 CONCLUSIONS AND FUTURE WORK

We have created an interactive system to help with lighting design in image synthesis by solving a restricted inverse lighting problem. The user paints an approximation of the desired result and the system computes light intensities and colors to match it. This approach can be more intuitive and easier to use than the usual direct edit–render cycle.

Given fixed geometry and a desired target, the problem of determining light intensities and colors can be solved in the least squares sense using a modified Gauss-Seidel algorithm. The method can be made more interactive by using incremental updates to the matrices and vectors involved in the solution process. Magnifying the effect of each brush stroke by increasing the weight of the affected vertices allows the user to make changes to the environment with relatively little effort.

Although they have received little attention in computer graphics, inverse lighting algorithms have great potential as design tools. Clearly there is much to do beyond automatic selection of light source intensities. Automatic light source placement would greatly increase the utility of the technique, but will require more elabo-

Figure 2: Design (left); best approximation (middle); ray tracing (right).

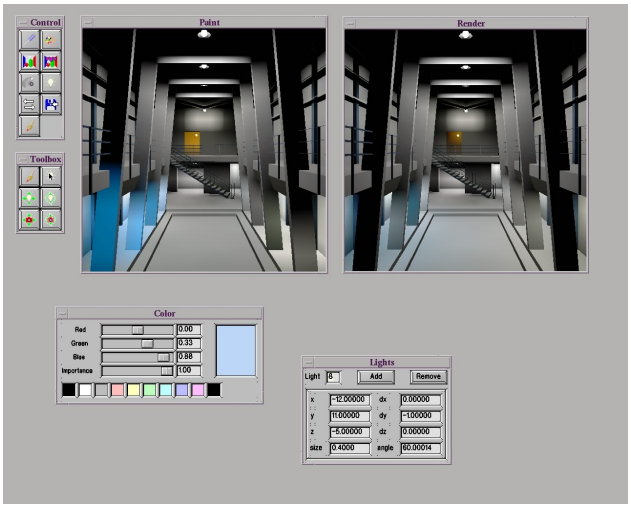

Figure 3: Interactive system.

rate optimization methods, as this requires solving non-linear constrained optimization problems.

Any rendering technique will work for determining the contributions from each of the lights. Our use of direct illumination only was motivated by a desire to allow interactive light placement. A more elaborate implementation might compute more accurate solutions for those lights that were unlikely to change position or distribution.

In order to make the system usable for lighting designers, some way of mapping screen intensities to physical units in the system must be found. Since the system is being driven by the user's perception of what is being painted, the lighting conditions of the user's environment must be accounted for, as well as the non-linearities of the monitor, the reproduction of color on the monitor, and most importantly, the extremely limited dynamic range of the monitor.

ACKNOWLEDGEMENTS

We would like to thank Jed Lengyel for his helpful comments and Kurk Dorsey and Suzanne Smits for their help assembling the paper. Much thanks to Matthew Bannister who created the model and the lighting designs. This work was supported by the NSF grant "Interactive Computer Graphics Input and Display Techniques" (CCR-8617880), and by the NSF/DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219). The authors gratefully acknowledge the generous equipment grant from Hewlett Packard Corporation on whose workstations the research was conducted.

REFERENCES

[1] Baltes, H. P., editor. *Inverse Source Problems in Optics*, Springer-Verlag, New York, 1978.

[2] Dorsey, Julie O'B., François X. Sillion, and Donald P. Greenberg. "Design and Simulation of Opera Lighting and Projection Effects," in *Computer Graphics*, 25(4), August 1991, pages 41–50.

[3] Evans, Ralph M. *Eye, Film, and Camera in Color Photography*, John Wiley & Sons, New York, 1959.

[4] Goral, Cindy M., Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. "Modeling the Interaction of Light Between Diffuse Surfaces," in *Computer Graphics*, 18(3), July 1984, pages 213–222.

[5] Hanrahan, Pat and Paul Haeberli. "Direct WYSIWYG Painting and Texturing on 3D Shapes," in *Computer Graphics*, 24(4), August 1990, pages 215–223.

[6] Lawson, Charles L. and Hanson Richard J. *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, 1974.

[7] Luenberger, David G. *Optimization by Vector Space Methods*, John Wiley & Sons, New York, 1969.

[8] Poulin, Pierre and Alain Fournier. "Lights from Highlights and Shadows," Proceedings of the 1992 Symposium on Interactive 3D Graphics, in *Computer Graphics*, April 1992, pages 31–38.

[9] Tumblin, Jack and Holly Rushmeier. "Tone Reproduction for Realistic Computer Generated Images," in Radiosity Course Notes of SIGGRAPH'91, ACM, August 1991, pages 229–257.

[10] Whitted, Turner. "An Improved Illumination Model for Shaded Display," *CACM*, 32(6), June 1980, pages 343–349.