Lighting with Paint

FABIO PELLACINI
Dartmouth College
and
FRANK BATTAGLIA, R. KEITH MORLEY, and ADAM FINKELSTEIN
Princeton University

Lighting is a fundamental aspect of computer cinematography that involves the placement and configuration of lights to establish mood and enhance storytelling. This process is labor intensive as artists repeatedly adjust the parameters of a large set of complex lights to achieve a desired effect. Typical lighting controls affect the final image indirectly, requiring a large number of trials to obtain a suitable result.

We present an interactive system wherein an artist paints desired lighting effects directly into the scene, and the computer solves for parameters that achieve the desired look. The artist can paint color, light shape, shadows, highlights, and reflections using a suite of tools designed for painting light. Our system matches these effects using a nonlinear optimizer made robust by a combination of initial estimates, system design, and user-guided optimization. In contrast, previous work on painting light has not permitted the lights to move, allowing for linear optimization but preventing its use in computer cinematography.

To demonstrate our approach we lit several scenes, mainly using a direct illumination renderer designed for computer animation, but also including two other rendering styles. We show that painting interfaces can quickly produce high quality lighting setups, easing the lighting artist's workflow.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism

General Terms: Algorithms, Human Factors

Additional Key Words and Phrases: Lighting, animation, rendering, optimization, painting

ACM Reference Format:

Pellacini, F., Battaglia, F., Morley, R. K., and Finkelstein, A. 2007. Lighting with Paint. ACM Trans. Graph. 26, 2, Article 9 (June 2007), 14 pages. DOI = 10.1145/1243980.1243983 http://doi.acm.org/10.1145/1243980.1243983

1. INTRODUCTION

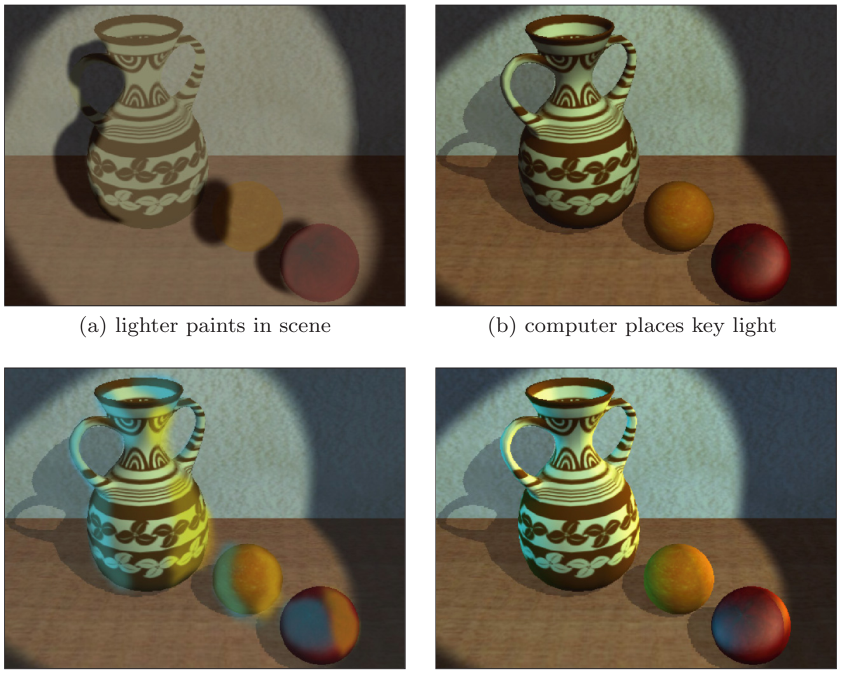
Lighting plays a crucial role in computer cinematography where, just as in live action film, it establishes mood, enhances storytelling, and frames the key elements of a shot [Alton 1949; Lowell 1992]. To achieve high quality lighting, artists called *lighters* often place tens or hundreds of carefully chosen lights, a very labor-intensive process, characterized by repeated adjustment of the parameters of each light in order to achieve the desired effect. The process is complicated by the fact that in typical shots the relationship between the parameters of the lights and the resulting visual effects is often unintuitive even for trained artists. This challenge is exacerbated by the increasing complexity of production shots as demonstrated by the growing size of the lighting crews required for feature-length computer animated movies.

To address this issue, we propose a workflow wherein the artist can directly paint the desired effects of lights into the scene using an interface similar to that of conventional painting programs such as Adobe Photoshop, leaving the task of finding the best values for the parameters of the given lighting model to the computer. As shown in the seminal text on cinematographic lighting, *Painting with Light* [Alton 1949], painting is a natural metaphor for lighting composition and such terms are used in practice to express desired lighting in movie production. This process, illustrated in Figure 1, not only allows for faster setup of lighting configurations, but often helps artists, trained or not, to achieve higher quality results by letting them work in a more familiar and intuitive interface.

To support this user interaction, our system casts the lighting problem as a high-dimensional nonlinear optimization whose goal is to find the best settings for each light parameter to match the painted input, the *target image*. The nonlinearity of the problem comes from the nature of lighting, for example, in the behavior of shadows or the responses of arbitrary surface shaders. To further complicate the problem, lighting setups used in computer cinematography are characterized by tens or hundreds of lights (each of which may have tens of parameters), requiring the optimization to be performed in a high-dimensional space.

One factor that largely alleviates these challenges stems from the observation that there is a trade-off between the quality of the solution and the complexity of the resulting lighting setup. On the one

The authors were partially supported by generous grants from Pixar Animation Studios and the NSF (IIS-0511965). The work was performed in part on equipment donated by NVidia Corporation; modeling and animation were performed on software donated by Alias Wavefront.
Authors' addresses: F. Pellacini; fabio@cs.dartmouth.edu; A. Finkelstein; af@cs.princeton.edu



(a) lighter paints in scene (b) computer places key light

(c) lighter paints more (d) final lit scene

Fig. 1. Lighting by paint: (a, b) the lighter paints to guide the starting configuration of a key light; (c, d) the artist paints more subtle lighting and the computer optimizes more lights to match.

hand, lighters are allowed to paint effects that cannot be achieved by any reasonable lighting configuration. On the other hand, while in most cases it would be possible to obtain a perfect match for the painting by creating a light for each pixel, this is probably not the intention of the artist but rather a result of inaccuracies (or abstractions) in the painted image. The workflow we propose allows the lighter to strike a desired balance between quality and complexity through several mechanisms. The lighter explicitly indicates to the system whether to add a new light or modify existing ones in order to match the painted image. After each optimization, if the desired result is not met, the lighter can add additional paint to steer the optimization in a given direction by modifying existing lights or to create a new light that will help match the desired effect. One benefit of this approach is that the lighter can retain control of the semantic meaning of lights (e.g., key light on Alice). Furthermore, he can focus on the desired goal (e.g., make this area more yellow) without worrying about either the specific parameters necessary to achieve that goal or the other side-effects of adjusting the parameters that achieve it.

Even with this simplifying assumption, basing an interactive system on high-dimensional nonlinear optimization may still seem like a dubious proposition. The major drawback of known methods is that they cannot guarantee finding global minima and tend to get stuck in local minima. Indeed, we have found that without careful treatment of the search problem, it is often the case that even sophisticated optimization algorithms will find local minima of very poor quality when adding a new light to the scene. We avoid such situations with a simple algorithm for estimating the initial configuration of the light that tends in practice to start near a good local minimum, requiring the optimizer to only finetune the solution. Furthermore, while we cannot guarantee a global minimum (as is the nature of nonlinear optimization), it is not necessary in our framework because the goal function is imprecise, and we are searching for any solution that matches the lighter's intuition in a way that is satisfactory to him.

While a general nonlinear optimization has the said drawbacks, it allows our framework to be applied in all lighting scenarios by treating a renderer as a black box. Thus our technique subsumes previous systems in which an artist can paint lighting effects. (Notable is the work of Schoeneman et al. [1993] that inspired our own by showing that painting is an effective way of specifying lighting. Their assumption that light positions are fixed allows the problem to be solved through linear optimization but prevents its use for lighting in computer animation.) To test this generality, we applied our technique to a direct illumination model used in computer animation, a global illumination renderer, and two nonphotorealistic styles. Within these renderers, the lighter was able to paint not only the general color of a lit object, but also shadow positions, light shape, and indirect lighting effects in mirrors and indirectly diffused environments.

The technical contributions of this work include (1) a suite of painting and compositing controls suitable for painting light into a scene with known geometry and surface properties, (2) a painting and optimization workflow suitable for lighting in CG production, and (3) a careful parameterization of the search space as well as a set of initial conditions that permits effective nonlinear optimization in this setting. While our efforts have emphasized use of this system in computer cinematography, some of these methods may transfer to other domains where lighting design is also known to be-difficult and important problem, for example, performance [Dorsey et al. 1991] or architecture [Schoeneman et al. 1993].

The remainder of the article is organized as follows. Section 2 places our method in the context of related work. Section 3 provides an overview of the process. Section 4 describes how the lighter paints as a way of describing his goals. Section 5 describes the heart of our system, the optimization method. Finally, Section 6 presents some results, while Section 7 gives conclusions and future work.

2. RELATED WORK

The system we present shares common goals with a number of previously described techniques. For example, one branch of the computer vision literature addresses the problem of inferring light configurations from shading and shadows in images, for example, the work of Wang and Samaras [2002]. The method described in Section 5 for finding a starting configuration for the optimization is informed by this literature. However, we present a fundamentally different approach to the problem based on user-guided optimization for three reasons. First, the lighting models used in animation typically have many more DOFs than just location and intensity, which are the focus of the vision effort. Second, the lighter generally provides input that is at once incomplete and inconsistent (imagine trying to paint accurate shadows coupled with consistent shading, for multiple lights without looking at a reference scene), whereas the vision research addresses a (certainly more difficult) challenge, that is, lack of knowledge of the underlying geometry and reflectance properties. Third, even an expert painter who could accurately simulate realistic lighting would generally choose not to do so for cinematic reasons [Barzel 1997], whereas computer vision uses models of physical reality.

Researchers have proposed techniques for easing the burden of lighting designers by giving them direct control of the effects of lights in the scene, for example, shadows and highlights. Poulin and Fournier [1992] presented a method wherein the designer manipulates lights by specifying highlights and by transforming shadow volumes in a wireframe view. A more recent technique developed by Pellacini et al. [2002] allows a lighter to drag shadows directly in an interactively rendered image and offers a constraint system helpful in scenes with multiple shadows. While these methods provide a remarkably intuitive interface for controlling the parameters that they affect (e.g., light position), they do not address the many different DOFs that we would like to control and can be confusing when adjusting the parameters of many lights.

Several researchers have investigated methods for optimizing lighting-related parameters in order to achieve various goals in rendered images. A survey of these methods, comparing their mathematical formulation, is presented in Patow and Pueyo [2003]. Here we review the methods more closely related to our work. The method of Kawai et al. [1993] optimized over the intensities and directions of a set of lights as well as surface reflectivities in order to best convey the subjective impression of certain scene qualities (e.g., pleasantness or privateness) expressed by users. Poulin and Fournier [1995] described a way to solve for for more high-level surface parameters (e.g., diffuse and specular reflection coefficients) for a given illumination model, consistent with a set of user-colored disks placed onto the model. Their work is orthogonal to ours but, at its heart, is a very similar nonlinear optimization based on painted lighting: they solve for surface properties where light parameters are known, whereas we solve the opposite problem. Marks et al. [1997] presented *design galleries*, which use a kind of user interface that is very different from the one we propose, in order to address a similar goal, namely, assisting an animator (or lighter) in the exploration of a very large parameter space to achieve a desired effect. Both Shacked and Lischinski [2001] and Gumhold [2002] tackled nonlinear lighting optimization, albeit for a different application domain than that addressed here: automatic lighting for novices based on perceptual qualities. An interesting approach to specifying optimization constraints is presented in Costa et al. [1999] where users can express arbitrary constraints for lights to be solved by the system. While this work is interesting, it is orthogonal to our own in that we are attempting to simplify the user interface as much as possible, while their work concentrate on complex constraints specification.

Perhaps most similar to our work are previous efforts that provide painting interfaces for expressing how models respond to light. Hanrahan and Haeberli [1990] first presented a WYSIWYG system for painting surface characteristics (e.g., textures) directly onto the model. Schoeneman et al. [1993] described a method where the user paints on a scene to be lit with global illumination, and the system solves for the intensities of a set of lights with known positions. Anrys et al. [2004] and Mohan et al. [2005] use a similar approach to relight real objects whose appearance is captured using image-based lighting techniques. Poulin et al. [1997] presented a sketching interface whereby the user expresses constraints on highlights and shadows (umbra and penumbra) for ellipsoid geometry, and the system solves for the appropriate position for a point or area light. Finally, Kalnins et al. [2002] presented a method for painting many aspects of stylization for NPR scenes, including a specific kind of hatching meant to suggest light or shadow.

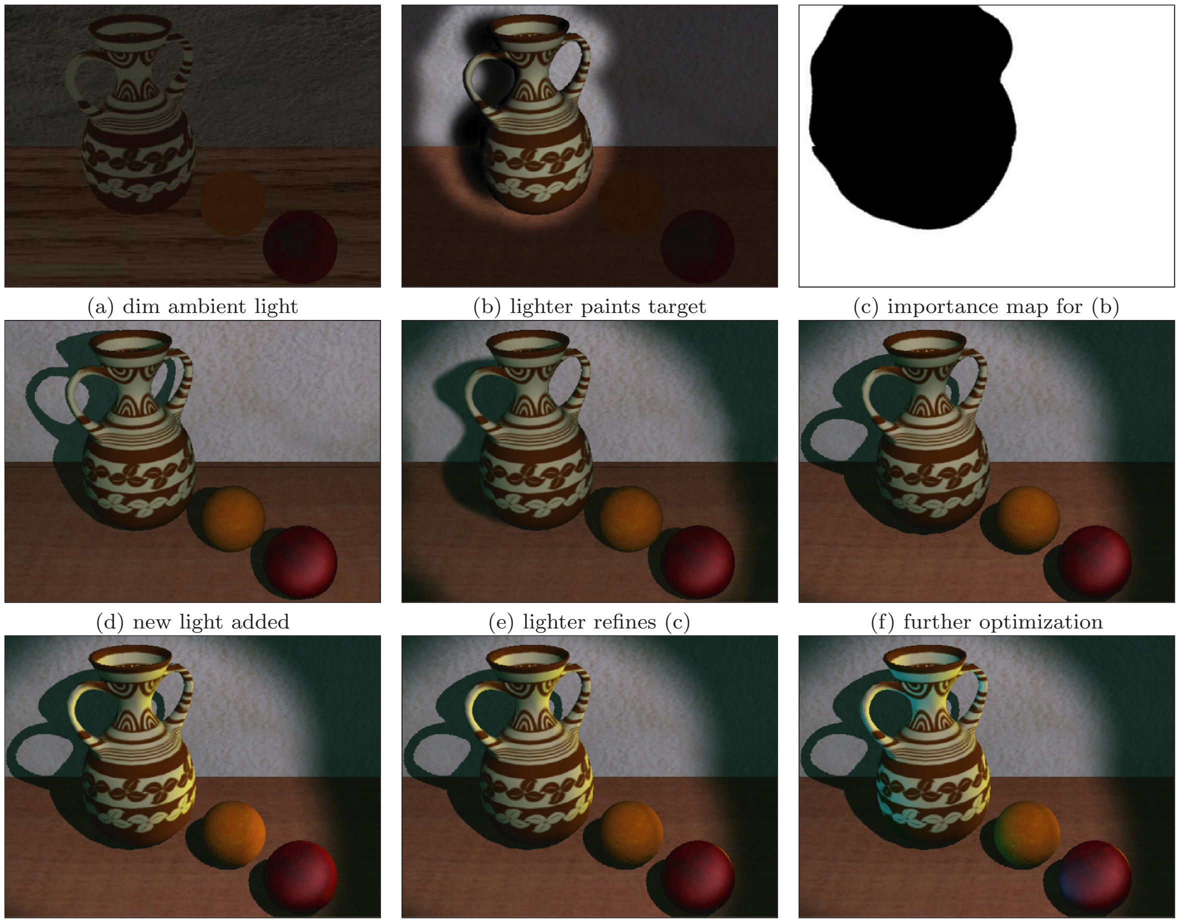
Largely inspired by these techniques, the system we propose is the first suitable for lighters to to be able to rig the complex configuration of lights, each with many DOFs, that are coupled with a complex model and arbitrary surface shaders in order to render a typical production shot. Specifically, the painting tools we describe in Section 4 are the first such tools appropriate for painting cinematic lighting over known geometry with known surface shaders. We also introduce a workflow in Section 3.2 that allows lighters to add lights one-by-one (giving him semantic control over each light) or to choose a set of lights to optimize together to match a particular effect. Finally, this article addresses a pair of challenges inherent in lighting for computer cinematography: the need to manage the many DOFs of cinematographic lights, and the nonlinear nature of the search space. Except for Shacked and Lischinski [2001], which targets a different application space, all previous light optimization systems made simplifying assumptions, for instance, fixed light positions, simple light models, or no shadows and reflections, any of which make the solution easier but limit its application in cinematography.

3. OVERVIEW

This section provides an overview of the workflow used by a lighter to set up lights using our system, as well as a high-level view of the algorithms working behind the scene.

3.1 Context

The role of the lighter is to make a visual composition: to frame the essential elements while expressing a mood appropriate to the shot using lighting that appears to be natural or consistent with the scene (e.g., near a window with daylight spilling in). In typical animations rendered with direct illumination, the lighter sets up many lights per-shot (tens or hundreds), each light having many degrees of freedom (DOFs). For example, a CG light designed to mimic the behavior of a live action barn light might have DOFs for position (3), orientation (2), color (3), distance cutoffs (4), distance falloffs (2), barn shape (3), shadow density (1), as well as more esoteric production-specific parameters [Barzel 1997]. (Our direct illumination renderer described in Section 6 implements this set of DOFs.)



(a) dim ambient light (b) lighter paints target (c) importance map for (b)
(d) new light added (e) lighter refines (c) (f) further optimization
(g) lighter paints yellow (h) yellow light added (i) final image with additional lights

Fig. 2. More detailed process by which a lighter sets up the lights in a shot (same as in Figure 1, but with different lights).

In finding a solution satisfactory to himself or the director, the lighter intuitively performs a high-dimensional search, for example, by adjusting an intensity of one light, moving another, changing a color on a third, then returning to the first to readjust it, and so forth. The purpose of the system presented in this article is to provide the lighter with more efficient tools for performing this search. The lighter guides the system through the search space by painting, and the computer peforms low-level optimization in response to the paint strokes.

3.2 Workflow

The workflow in our system can be illustrated by the following high-level example, shown in Figure 2. Note that some actions labeled painting in images (b), (d), (f), and (h) may not look like painting because of the use of sophisticated brush controls described in Section 4 and highlighted in Figure 3.

(a) The lighter first loads the scene, which is by default lit by a dim ambient light so that the lighter can see the model.
(b) The lighter paints some light on the jug as well as a shadow on the wall behind.
(c) Implicitly by painting in some areas, the lighter has set up an importance map described in Section 4. This image tells the optimizer how much relative weight to give to different areas of the target image (black = important).
(d) The computer adds a new light and optimizes its parameters with the goal of matching what the lighter has painted, emphasizing of course the important areas.
(e) The lighter decides to modify the shape of the newly added light as well as the shape of the shadow of the jug. He paints to refine the image from (d).
(f) The computer further optimizes the light parameters to better match the lighter's refined target image.
(g) The lighter paints new yellow light from the right.
(h) The computer adds a light and optimizes to match it.
(i) The lighter paints the effect of a blue light from the left.
(j) The computer optimizes to match it.

This process is designed to provide the lighter with controls that correspond to a natural way to think about lighting, that of painting with light [Schoeneman et al. 1993] after [Alton 1949]. The lighter thinks in terms of composition in the frame, and the computer performs the messy part of the high-dimensional search that would otherwise have to be performed by the lighter repeatedly tweaking various parameters. Note, however, that the lighter has

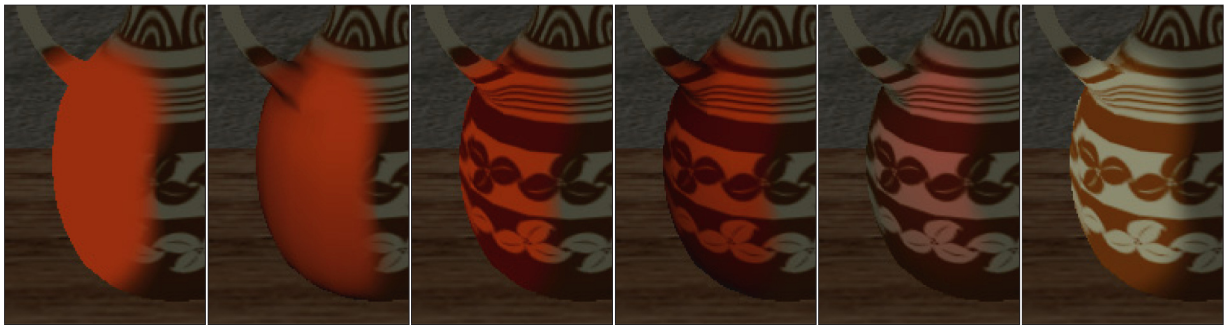

(a) $\gamma = 0 \quad \tau = 0$
(b) $\gamma = 1 \; \tau = 0$
(c) $\gamma = 0 \quad \tau = 1$
(d) $\gamma = 1 \; \tau = 1$
(e) add light
(f) brighten

Fig. 3. Similar strokes drawn in various painting modes. Images (a–e) have the same color, brush size, and full opacity. Images (b, d, e) reveal a gradient. Images (c–e) respect the model's diffuse texture. The brush in (e) has the settings as in (d) except that it cannot darken in any channel. In (f), the brush brightens the image.

explicit control of when new lights are added, for two reasons. First, the computer does not have to perform a discrete optimization over the number of lights in conjunction with what is already a difficult continuous optimization over their parameters. Second, the resulting lighting rig corresponds to something that the lighter intended (the lights have explicit meanings to the lighter, e.g., key light on Alice or rim light on vase) and is therefore easier to understand and manipulate later.

In addition to explicit control of when to add new lights, the lighter also may choose which parameters of which lights are free to change during optimization. The main reasoning behind this design decision is that the lighter may be pleased with the current qualities of some lights and should have the ability to lock them in place. However, it also has two other benefits. First, the lighter may be attempting to achieve a particular effect and has a good idea of what parameters need to be adjusted to achieve it. Second, since the dimensionality of the search affects time (and success) of convergence, the ability to choose which parameters to optimize allows the lighter a way to guide convergence. Finally, it is the lighter who decides when the optimization has converged to satisfaction. As the optimizer searches for a good lighting configuration, our application shows a rendering of either the current trial or the best configuration found so far (which we believe lighters would generally prefer to view).

To facilitate the workflow described in this section, two questions must be addressed. First, how can the lighter paint images easily and with sufficient accuracy to show the desired lighting effects? Second, how can the computer find a sufficiently good match to the desired lighting? These questions are addressed in Sections 4 and 5.

4. PAINTING

This section describes the painting interface by which the lighter expresses how he would like to set up lights in the scene. When the scene is first loaded, it is rendered with a dim, ambient light so that the lighter can see the objects in the scene. To change the look of the scene, the lighter paints directly in the image, and then starts the optimizer which tries to match the painting. We have found that attempting to express lighting in detailed scenes using only traditional compositing brushes can be cumbersome. Therefore, we have augmented traditional tools with a set of specialized brushes that can leverage knowledge of the underlying geometry to help the lighter easily and accurately paint light into a scene.

4.1 Brushes

Before painting a stroke, the lighter chooses five brush qualities: color c, diameter d, opacity α, texture strength τ, and gradient γ. Opacity has the usual meaning for compositing and is implicitly multiplied by a gaussian of width d pixels centered at the mouse position. Texture strength provides control over how much the painted color is modulated by the diffuse texture color of the objects being painted (for example, the pattern on the vase in Figure 1). Varying this parameter allows for the painting of specular effects. Suppose p_b is the color of a pixel before painting and c_d is the diffuse texture color of the object at that pixel, then the painted color $c_p = (1-\tau)c + \tau(c \cdot c_d)$. Colors have three color channels and the math may be performed for each channel independently.

Rather than requiring the lighter to paint with a flat, constant color, we also provide a gradient tool similar to the tools in Adobe Photoshop or Gimp. In commercial painting programs such tools can be used to fade strokes as though the paint runs out as the brush is dragged along the canvas, allowing the painter to create smooth color gradients on the painted object (Figure 3(b)). In our system, we know the underlying geometry and have found that a useful model is to record the surface normal $\hat{n}_{peak}$ at the start of a stroke, and then modulate the painted color by how much the normal $\hat{n}$ changes during the stroke. $c_\gamma = c_p((1-\gamma) + \gamma\ (\hat{n} \cdot \hat{n}_{peak}))$. When γ is 0, colors are left unaffected; when it is 1, then the effect is that of a diffuse directional light pointing at -$\hat{n}_{peak}$; of course $0 < \gamma < 1$ yields linear interpolation. With a modifier key the lighter can paint several strokes using the same $\hat{n}_{peak}$. Using another modifier key, the lighter can also fade away from a specular peak (rather than diffuse); we set $\hat{n}_{peak}$ to the view vector reflected about the normal.

The color after compositing is given by $c_c = (1-\alpha)p_b + \alpha c_\gamma$. Obviously, if the lighter chooses to add a light to the scene without modifying the existing lights, the new light can only brighten (not darken) the image. For such cases, we provide a painting mode called *add light* that prevents darkening by the painted color, whereupon the pixel after painting $p_f = max(p_b, c_c)$.

4.2 Importance

The target image has a fourth channel called the *importance map* (e.g., Figure 2(c)) that allows the lighter to express for the optimizer the relative weights of different areas of the image. The optimization must balance between competing goals (e.g., brighten the orange but don't change the table), and the importance map provides a control

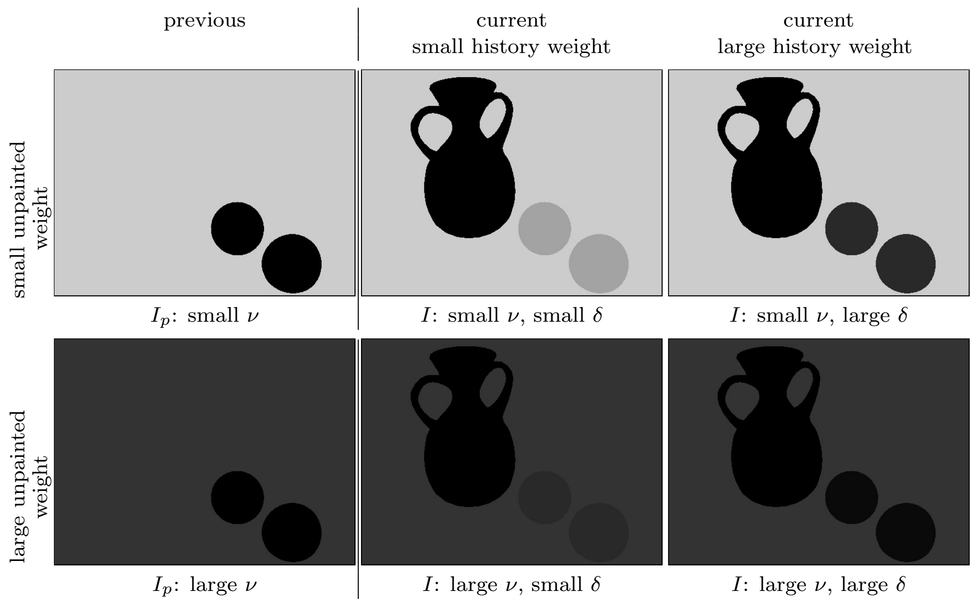

previous
current small history weight
current large history weight
small unpainted weight
large unpainted weight
I_p: small ν
I: small ν, small δ
I: small ν, large δ
I_p: large ν
I: large ν, small δ
I: large ν, large δ

Fig. 4. Importance maps let the lighter to express the relative weights of different areas of the target image (white = 0; black = 1). Left: Fruit were painted in the past. Middle: Jug painted now with low history weight δ. Right: Jug painted now with high history weight δ. Top: Background remains more important given the large background weight ν. Bottom: Background remains less important given the small background weight ν.

whereby the lighter can express the relative importance of these goals. The lighter can paint the importance map explicitly, but this can be cumbersome. Therefore, we also provide a way to express it implicitly by setting two controls and painting normally in the color channels. The first control, ν (unpainted), says how much weight to give to pixels that are left unpainted in the target image (and should therefore change as little as possible during optimization). The second control, δ (history), says how much weight to give to whatever the lighter painted in the past. The effects of these two parameters are demonstrated in Figure 4. Let I be the importance map for the current target image and I_p be the importance map from previous target image set initially to ν. We define the previous target image as the image the user obtained after one optimization, which might include multiple strokes. Pixels in I are 1 if the lighter painted them in this target image. If not, $I = (1 - \delta)\nu + \delta I_p$. The benefit of this implicit specification is that the lighter can express the importance map with two simple controls, both of which have an intuitive meaning. However, in cases where the lighter wants explicit control, he paints directly into the importance map using the implicit version as a starting point.

The GUI also allows the lighter to pick a specific object on which the paint strokes will be clipped (rather than falling on other items across silhouettes). The system supports painting modes for brightening, darkening, or for copying from an image that has been modified in an external program (allowing, e.g., histogram equalization, posterizing, etc.) We also provide a *shadow brush* that can paint and erase shadows when the lighter wishes to modify a light or set of lights. The shadow brush and eraser work by copying pixels from a pair of rendered images, one that contains the scene lit with all the modifiable lights turned off and the other containing all the lights turned on but the shadows turned off. Finally, there is also a paint bucket brush. These tools may be used in combination, allowing for example, for uniform brightening of a specific object with one mouse click. See Figure 3 and for examples of various painting effects.

4.3 Discussion

Since the goal of our interface is to reduce the complexity of the lighting design task, it might appear that introducing multiple parameters and painting tools in our lighting interface could confuse users. First, when comparing our tools with similar retouching tools for digital imaging, our interface should feel at once familiar and simpler in the number of parameters used since we designed our tools to be the equivalent of the traditional ones when applied to lighting design. Furthermore, the added controls for textures, importance, and gradient are only meant to reduce the number of strokes lighters have to apply; it is possible to use our interface without ever needing to specify any of those parameters. We have also observed that it is typical not to modify those parameters often.

5. OPTIMIZATION

At the heart of our system is an optimization algorithm whose purpose is to unburden the lighter from having to optimize the light parameters manually. The goal function for the optimization is the weighted average of the difference between pixels i in the target image T and the rendered image R:

$$\epsilon = \left(\sum_i I_i \cdot ||T_i - R_i||_2\right) \Big/ \left(\sum_i I_i\right),$$

where I is the importance map. Several factors make this a difficult function to optimize. First, it is nonlinear in the parameters of the lights. Second, to evaluate ϵ, we render R_i and subtract it from T_i, which can be expensive in the inner loop of an optimization. Third, if we are optimizing many parameters or many lights, the search takes place in high-dimension. Fourth, there is no general strategy (other than sampling) to determine the local gradients of the goal function. Fifth, our search space has large plateaus in which the optimizer can get lost, for example, where a distant spotlight is pointing away from the scene or where the light is behind or inside the objects in the scene. (In these situations, even a large change in the parameters of the light may not have any impact on the rendered image.) Our initial experiments with a simple hill-climbing strategy showed that it is easy to become stuck in such plateaus whenever the starting configuration was worse than the plateau. For example, if a distant spotlight starts by illuminating the wrong side of a painted object, the optimizer will quickly find that it can improve matters by shining the light away from the scene entirely.

Our system is designed to work with any optimization algorithm, but, as a practical matter, there are few optimizers appropriate for the problem we have described Press et al. [1995]. For example, a workhorse nonlinear optimizer like simulated annealing requires many evaluations of the goal function while the annealing schedule cools in order to converge on a good solution. Likewise, methods such as conjugate gradient would require us to compute partial derivatives by sampling the goal function in each dimension for each step of the optimization. Because evaluation of our goal function is expensive, requiring a full rendering of the scene, we cannot afford such methods for our interactive system. Finally, by exploiting known linearities in some parameters of a specific lighting model, it might be possible to use a combination of linear and nonlinear optimization methods. However, this would require making assumptions about the lighting model, which we prefer in this initial project to treat as a black box (allowing the use of alternate models as shown in Figures 8 and 10).

In our experiments we found that the nonlinear simplex method of Nelder and Mead [1965] and Press et al. [1995] works well provided that (1) the search space is parameterized well, and (2) it starts with a decent configuration. (These caveats are addressed later.) For a search in n dimensions, this method stores a set of $n + 1$ configurations representing a simplex in the search space. At every step, the optimizer considers a set of possible configuration changes for the *worst* vertex (the one with the worst value in the goal function), such as reflecting the vertex through the opposite face in the simplex. Each considered change requires one evaluation of the goal function. If no possible improvements are found, the simplex shrinks by uniform scaling toward the best vertex. When the simplex has collapsed to a very small size, it has found a local minimum. This solution is typically good enough and the lighter halts the optimizer there. However, we have observed that sometimes it helps to restart the optimization from this local minimum to see if it can find a better local minimum by growing the simplex to its original size and beginning the search from there. Therefore, our application repeatedly restarts the optimizer until the lighter chooses to halt it.

5.1 Parameterization and Constraints

We have found it beneficial to map all parameters' canonical ranges to [0,1] for the purposes of optimization. For example, angles, color intensities, and shadow density all lie in this range. This way when the optimizer takes a uniform small step in any dimension, we do not get wildly different impact simply from the dimensions having different scale. For some parameters, values outside this range are meaningless to the renderer (e.g., a light with negative color intensity or negative spotlight angle). Thus, in addition to the difference ϵ between the target and rendered images, we add terms to the goal function that impose (soft) constraints on the lights, ensuring that the parameters remain in their meaningful range as follows. For a parameter x that should be in the range [0,1], we add to ϵ the value $max(0, (2x-1)^2-1)$. This penalty function is zero when x is in the valid range and climbs steeply outside this range. In principle, these constraints might also be addressed using the constrained simplex method (called COBYLA) of Powell [1994], although it is believed to converge slowly for highly-nonlinear functions.

5.2 Starting Configuration

Key to finding a good solution during optimization is starting the search from a good configuration. For example, if the light starts below the floor on which it should cast light, it is possible for the optimizer to get stuck and never figure out where it needs to move the light to achieve the appropriate effect, especially if it is optimizing over many parameters at once. When the lighter is not adding a new light, we start with the current configuration of lights, based on the rationale that the lighter is steering the optimization by painting from the current state. On the other hand, a starting configuration for a new light is computed based on the set of pixels i, where color c_i is painted over background b_i by the lighter. Each pixel has associated with it a point p_i and normal $\hat{n}_i$ in 3D as well as a diffuse color k_i. We begin by finding the *hot spot*, the center of illumination by the light, taken to be the average p_i of the pixels that were lightened by painting. Next we find a direction for the light with the rationale that it is known from computer vision that for nonlocal lighting effects there is an ambiguity between the distance and intensity of a light. Assuming Lambertian shading and light with intensity L and direction $\hat{d}$, the following property should hold:

$$c_i - b_i = k_i L(\hat{n}_i \cdot \hat{d}).$$

In this equation, the only unknowns are L and $\hat{d}$. Providing that more than a few pixels are painted, this system of equations is overconstrained. We find the least-squares best solution to this system in each of the three color channels, yielding estimates for the intensity in each channel as well as three directions. We clamp the color in each channel and then take the light direction to be the average direction, weighted by the intensity in each channel. (If the equations are ill-conditioned we simply use the average painted color and average surface normal.) Finally, we set the light position to be the model diameter away from the hot spot in the direction $-\hat{d}$. If the new light is a spotlight, we find the minimum angle that would cover the bounding box of the painted pixels.

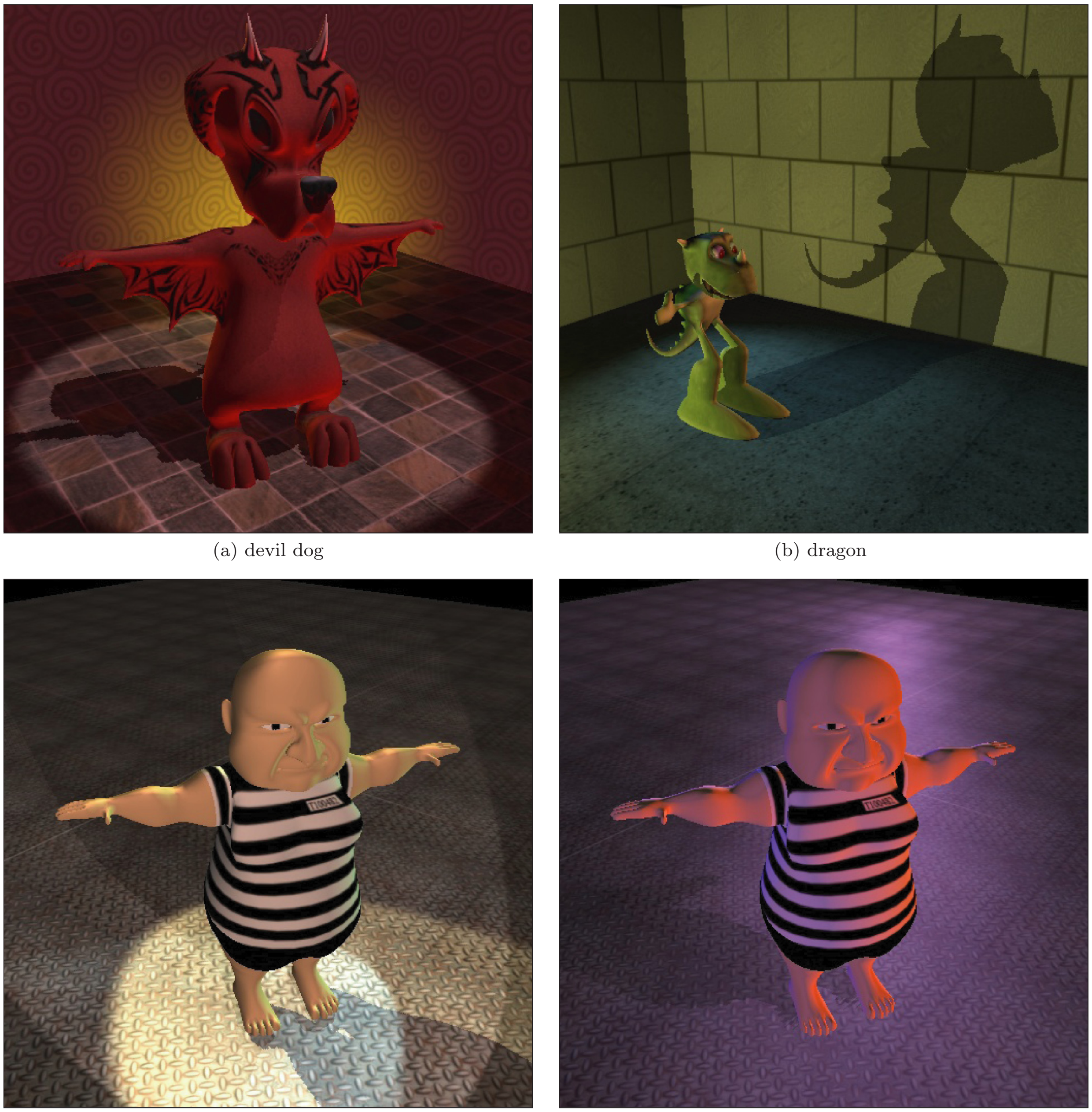
5.3 Multiresolution

In cases where there is an obvious way to trade off quality versus time during rendering, it makes sense to run the optimization with a fast, low-quality renderer at first and then rachet up to slow, high-quality renderings near convergence. We have explored this method with the GI renderer (Section 6.1), roughly halving the overall running time of the example shown in Figure 10. With the GI renderer, it is easy to make the trade-off by simply starting with low resolution (64×64), and then, when the optimizer converges to a local minimum, jump up to a higher resolution, and repeat (in this case, up through 256×256). This example is discussed further in Section 6.

6. RESULTS

Figure 5 shows some example characters lit with our system. Each example uses 6 or 7 lights and took between 45 and 90 minutes to light by painting. The bulk of this time was spent experimenting with various lighting effects (not waiting for optimization). When adding or modifying lights, the rendering and optimization times vary with the problem. Some tasks converge in a matter of just a few seconds, while for others it might take tens of seconds.

Most of our efforts have focused on a hardware-based renderer for realistic direct illumination by a cinematic light. The illumination model, which has the 18 DOFs mentioned in Section 3.1, is



(a) devil dog

(b) dragon

(c) prisoner 1

(d) prisoner 2

Fig. 5. Three example characters lit with our system. The third is shown with two differerent lighting rigs to suggest the dramatic range available through lighting.

similar to the one described by Barzel [1997]. This model can be implemented in Renderman or other conventional renderers used in production settings. Such renderers generally are not optimized for real-time rendering. However, with modern graphics hardware, it is possible to implement an excellent approximation that achieves many frames per-second [Gershbein and Hanrahan 2000; Pellacini and Vidimce 2004; Pellacini et al. 2005]. For each pixel in the image, our system caches all information about the visible point that are required for relighting (i.e. position, normal, and material parameters) in a deep framebuffer data structure. This gives it scalability with geometric complexity in that the main camera visibility is resolved only once at startup. Shadows are computed using standard shadow maps implemented in hardware. In CG animation, shadows are often deliberately simplified through a simple trick: one light can use the shadow map of another light. Therefore, while by default our renderer uses a separate shadow map for each light, we have also implemented a mode in which the lighter may choose to have all lights use the shadow map of a single key light. This effect

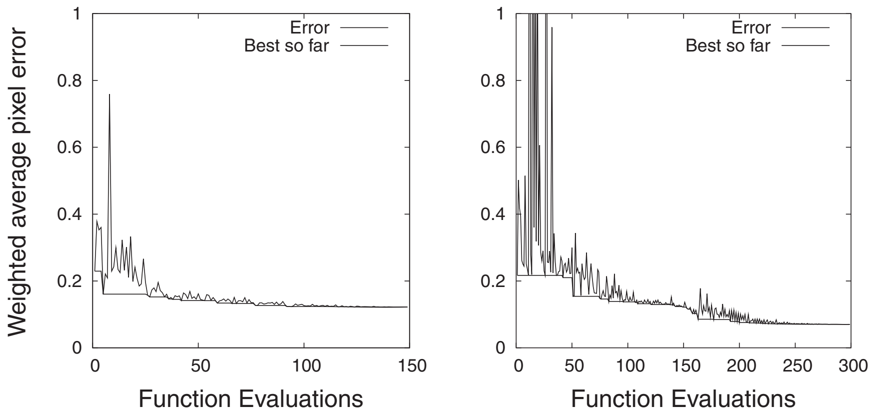


Fig. 6. Convergence rates for the images in Figures 2(d) and 2(f), each requiring 85ms per-evaluation for a total time to convergence of about 13 and 25 seconds, respectively, while optimizing over 7 and 12 parameters. For some evaluations, ϵ can exceed 1.0 if the optimizer tests illegal configurations (e.g., negative colors).

Table I. Convergence Rate as the Number of Varying DOFs Increases

lights	DOFs	evals	time
1	5	100	8s
1	18	340	27s
2	30	630	63s
4	51	2,640	194s

is obvious in Figure 5(c). Finally, the image difference used for the error function of the optimizer is also computed on the graphics card, further accelerating the inner loop of optimization. While our system is implemented using graphics hardware, the renderer and optimization algorithm perform all computation in floating-point and at no-point clamp their values in the [0, 1] range. This lets us support realistic lighting, including shaded values greater than 1 and arbitrary tone-mapping functions.

The direct illumination renderer runs at interactive frame rates on a 3Ghz P4 CPU coupled with an nVidia 6800GT graphics card. When shadows are rendered, typical frame rates are 10fps, whereas without shadows, the system often runs above 40fps because all of the geometry has been sampled per-pixel. Due to per-pixel point sampling, even the David model shown in Figure 7, which has 250K faces, renders at interactive rates, provided that shadows are not enabled for the light rendered. Note though that rendering is in the inner loop of an optimization process that may require many iterations. Time to convergence during optimization depends heavily on the particular problem because the the number of evaluations will vary with the input and the number of DOFs that are allowed to vary as well as the starting configuration. Typical convergence times range from a few seconds to a minute for finding the color, position, direction, and shape parameters (12 DOFs) for a new light. Two examples of convergence are shown in Figure 6. These plots are characteristic of optimizations for lights that have shape and shadows. They tend to converge more slowly than the parameters of plain point lights, both because rendering shadow maps slows the renderer (making each iteration take more time) and because the search space is more nonlinear, generally requiring more interations of the optimizer.

Table I reports on a simple experiment providing a sense of how quickly (iterations and time in seconds) the optimizer converges as the number of DOFs increases. Starting with the lighting rig shown in Figure 5(d), we randomly changed several of the light parameters by between 10% and 50% of their useful range. Next we measured the change ϵ to the resulting image. Using the original image (Figure 5(d)) as a target and unlocking only the perturbed DOFs, we ran the optimizer until the error fell to 0.05ϵ. (Note that the error will never fall to zero due to numerical imprecision in rendering and optimization.) An image with error 0.05ϵ is typically visually indistinguishable from the target. This represents a conservative measurement relative to use in practice wherein the target image is crudely painted and therefore not achievable (nor desired).

6.1 Extensions

6.1.1 *Light Removal.* One possible application of the optimization methods described here is targeted removal of specific lights in a scene without removing the effects of the deleted lights. This strategy could be used, for example, to reduce the complexity of a lighting rig or to reduce rendering times (which tend to scale with the number of lights). In Figure 7, (a) the model is shown lit with seven lights (and no shadows) set up in our system, while image (b) shows the effect of only the strongest of them, light #1. We removed light #1 and asked the optimizer to attempt to match (a) with the remaining six lights, varying positions, directions, and colors (36 DOFs in total). After five minutes, the optimizer converged on the configuration shown in image (c). Image (d) shows a 2D projection of the 3D locations of the lights relative to the head, suggesting both that this procedure might be difficult to achieve by hand and that the light locations in (a) were not obviously redundant. Note that we deliberately chose the strongest light in this experiment to avoid concerns that the effect of the removed light might have been minimal. However, in so doing, we gave the optimization the worst possible starting configuration for its search. More thorough algorithms for automatic light removal would probably want to try remove weaker lights to begin.

6.1.2 *Nonphotorealistic Rendering.* Our method also works with more stylized renderers. We have implemented a few simple hardware-based NPR shaders. One style implements 'toon shading by thresholding the angle between the light and the surface normal. The other style shades a parameterized surface using hatching strokes, based on the method of Praun et al. [2001]. Finally, inspired by the method of Raskar and Cohen [1999], we emphasize

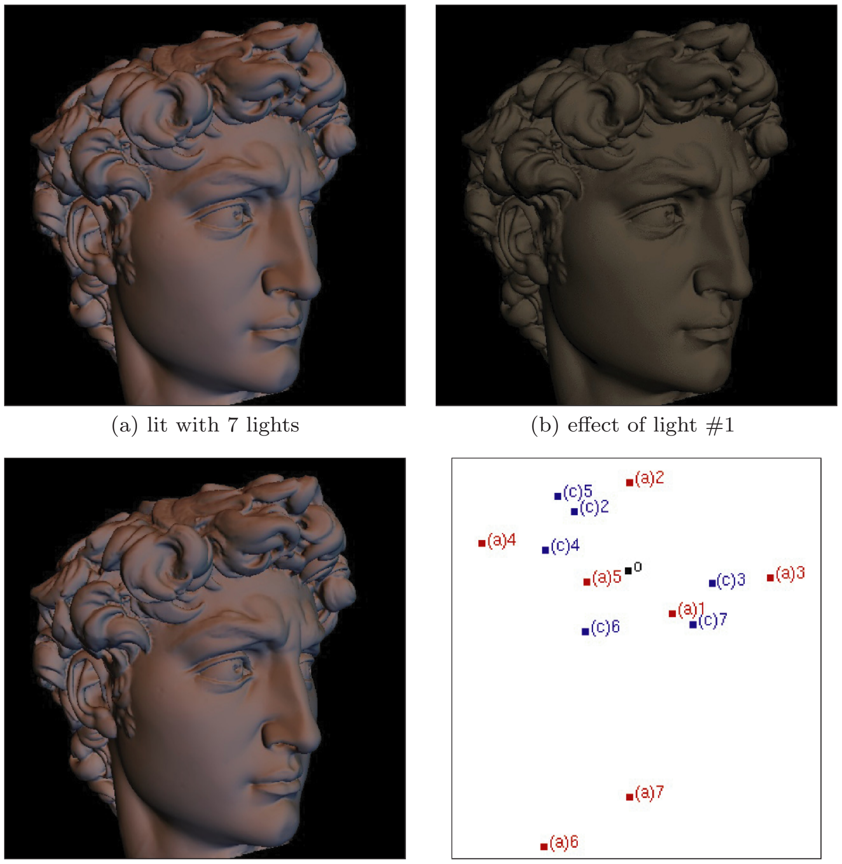


Fig. 7. David's head (a) with 7 lights; (b) effect of light #1 only; (c) with light #1 removed, the other 6 lights optimized to match (a); (d) light locations in images (a red) and (c blue) with o = the model.

silhouettes by drawing in black a slightly inflated version of the model, displaced slightly behind the true model.

The NPR examples shown in Figure 8 also show that this framework can be used to set up lights with very different types of renderers. These pixel-shader-based renderers run at interactive frame rates for simple models. In the hatching example, we perform the error calculation at a slightly blurred version of the target image because in such style the goal is to make all the pixels either black or white (whereas none of the pixels in the target may be those colors). In the examples shown, we set up a single light to be consistent with the aesthetic of those styles. In the case of the 'toon shader, the optimizer also adjusts the color of the bright regions as well as the modulation of the dark regions.

6.1.3 *Global Illumination.* Most CG animation is rendered using direct illumination. To achieve complex, naturalistic effects, the lighter creates many lights that simulate the effect of secondary bounces. A few studios are beginning to render animations using a global illumination (GI) algorithms in order to attain subtle lighting effects while reducing the number of lights.[1] To demonstrate that our method can work with GI, we have implemented a noninteractive version of our program based on a (relatively slow) publicly available path tracer [Shirley and Morley 2003].

In the GI example shown in Figure 10, we painted out any importance from the ceiling in the importance map, in order to allow the light to move. (It is not generally the case in computer animation that a lighter would want to position a light that is actually seen in the rendered image.) The light was constrained to move on the ceiling, and we optimized over 4 parameters x, y, size, intensity. The importance map used is shown in Figure 10(c). In this example, the time to convergence for the multiresolution approach is 21 minutes as opposed to 48 minutes for optimizing at the full resolution only. For GI, the optimization time is dominated by rendering, and most of the overall time is spent optimizing at the highest resolution. The benefit of the multiresolution approach is that it gives the highest resolution a good starting point, thereby accelerating time to convergence. We considered using this multiresolution, approach for the hardware renderer as well but found that our render time for

[1]Unfortunately, this strategy can hamper artistic control in two ways. First, GI can increase the rendering cost (thereby reducing interactivity). Second, GI eliminates many of the DOFs used by the lighter to make the scene look the way he likes. Often a lighting director will make a request like "warm tones over here", which can be difficult to achieve with traditional GI, particularly because the adjustment of a light to achieve a specific local effect may also impact other areas in the scene.

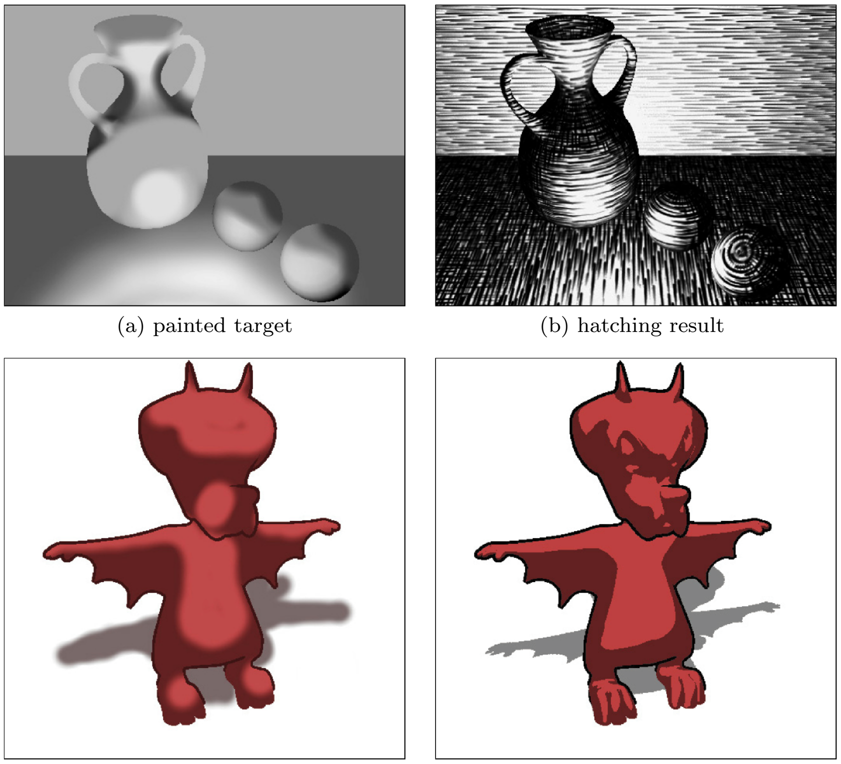

(a) painted target (b) hatching result (c) painted target (d) ’toon result

Fig. 8. Two examples of nonphotorealistic rendering styles.

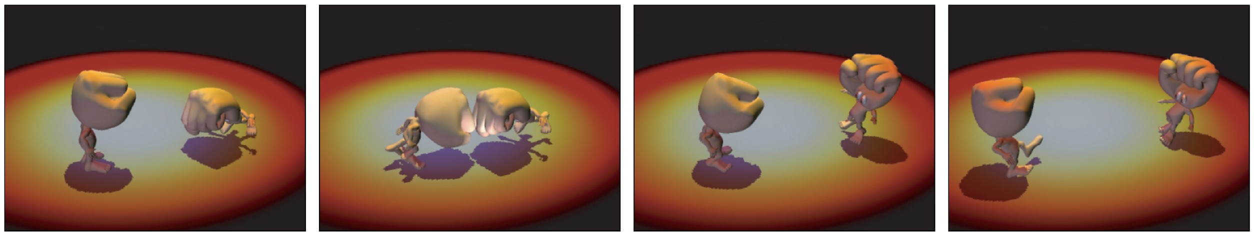


Fig. 9. Four frames from an animated shot. The lighter set up the lights by simultaneously optimizing painted effects in the middle two frames. The other frames in the sequence were rendered using the same lighting setup.

hardware is not shading-rate bound so reducing resolution does not help substantially. Nonetheless, other methods for trading quality for performance might help the hardware renderers.

6.1.4 *Animation.* Our system can be used to light animated sequences following the established production workflow of lighting different frames independently and interpolating the lighting setup by keyframing. Our system can accommodate this by creating a set of lights whose parameters are independently optimized for two separate frames. Even for this case, it is important to build the lighting configuration one light at a time since this helps to make sure that the interpolated resulting animation is what the user wants. We suggest that the user add a new light and optimize its position in all the keyframes before mocing to another one. If the interpolated configuration is not what the user desires, the lighter can add more keyframes to guide the interpolation of the configuration. This is the same established workflow for keyframe-animated lighting animation that is followed when using traditional tools.

A more complex and less frequent example is if we want to illuminate our sequence with lights that have the same parameters across multiple frames. Our system allows for these kinds of optimization by letting the lighter paint on multiple separate frames, whose combined error is minimized. An example of this process is illustrated in Figure 9 where the two internal frames were painted to obtain the given sequence. Interestingly the same mechanism used for animation can be used to light scenes that have to be viewed from different points of views, guaranteeing that the scene will look interesting from each. While we covered the most typical case and its straightforward extension, we leave it for future work to establish whether this is the correct semantic for these kind of situations or whether the energy function formulation should be more complex.

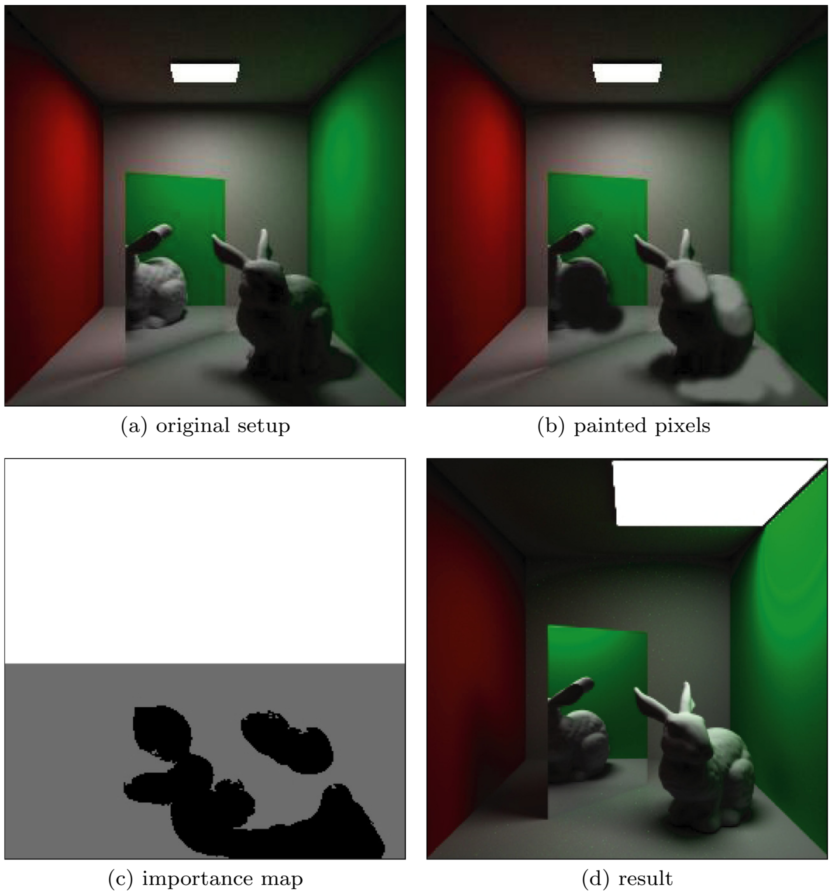


Fig. 10. Global illumination. Starting with (a) the lighter painted on the image—even in the mirror—to make (b). The importance map (c) tells the optimizer not to consider the upper part of the room during optimization, resulting in final rendered image (d).

6.2 Discussion

The experiments reported in this article were lit by novice lighters (namely, the authors of the article), demonstrating that this kind of interface makes it possible for nonexperts to achieve high quality lighting. None of these users would have been able to achieve the same quality results using a standard interface in the same timeframe.

While our experiments have focused on setting up lights solely through a painting interface, there are surely some tasks that are more easily performed with a more traditional direct manipulation interface. We intend for the system we present to augment rather than replace those tools, giving the lighter the option to use whichever tools are most appropriate for the task.

With the exceptions of the experiments shown in Figures 7 and 10, the lighter rarely waited for more than one minute for an optimization. It seems that any optimization that might take longer than that to converge on a local minimum wears out the patience of the lighter who tends to stop the optimization and continue work from there. In cases where the optimization simply won't converge on anything useful, it often turns out that, when the user thinks about what he has painted in the painting window and the importance map, he realizes that he has asked for something terribly inconsistent.

The main limitations of this framework stem from the two fundamental aspects of our system, namely, painting and optimization. First, there is often no way to know what the lighter meant by painting something. The goal function ϵ does not capture the artist's intent nor does it consider many aspects of perception; all we know is that when ϵ is low, then the rendered image visually matches the target image. Unfortunately, if ϵ is not low, we do not know if the reason is that the match is perceptually good but is not well captured by ϵ, or if the optimizer did a bad job of finding a minimum, or if there is no way to solve for what the lighter painted. Because the problem is nonlinear, we can make no guarantee about finding a global minimum during optimization. There is a subtle interconnection between these statements because if we could guarantee a global minimum, then we could know if what was painted was impossible to achieve through optimization.

Thanks to the deep framebuffer formulation, our system scales well with geometric complexity even in production environments as shown in Pellacini et al. [2005]. If more complex scenes are required, the addition of geometry LODs for shadows and more efficient culling will increase the performance significantly. We found that increasing the number of degrees of freedom into the hundreds will slow down the system significantly. This effect will be most acute when each DOF needs to change significantly in order to accommodate the required lighting change. An unfortunate scenario would be setting up a scene with a large number of lights and then repainting most of the scene. However, while possible, this does not match the workflow in production environments. Instead, in the proposed lighting workflow described in Section 3.2, lighting rigs are typically constructed one light at a time. When changes are required, only a small number of DOFs are allowed to change in order for lighters to maintain the semantic essence of each light. In this sense, the cases reported in Section 6 are designed to stress our system, indicating that it remains a viable solution.

While we have only tested our system with nontrained users, we hope that a future version of our tool will be able to simplify the lighting process for trained artists. Our system only assumes that the user has an understanding of lighting, a good aesthetic sense, and that, to some extent, they know how to paint. These are all qualities that a trained lighters would have. We hope that our system can enhance the workflow of such artists by accelerating the lighting process (and perhaps make it more enjoyable) as they will spend less time worrying about how to achieve the effects they want and thereby focus their time designing those effects. The main ideas in this new workflow were suggested to us by professional lighting experts working in production environments [Kalache 2004]. However, our method has yet to be implemented in a production system, and as such, it is not yet possible to measure its effectiveness in such real-world situation.

7. CONCLUSIONS AND FUTURE WORK

This article presents a framework where light designers directly paint the desired effect of lights in the environment, while the system finds the best settings of the parameters of these lights required to achieve the wanted look. Our system achieves these results by casting the problem as a high-dimensional nonlinear optimization that is solved by a nonlinear optimizer made robust by a combination of initial estimates, system design, and user-guided optimization. Using our framework, we lit various scenes and animations rendered with different rendering styles (ranging from a direct illumination model used in computer cinematography to global illumination to nonphotorealistic styles) demonstrating that painting interfaces for lighting design can be used to quickly produce high-quality lighting setups, providing major benefits in an artists workflow.

This research suggests several areas for future work, including the following.

Realistic Lighting Design. We would like to investigate the use of such methods for architectural and lighting engineering applications. In these applications some DOFs of lights are more constrained (e.g., lights generally have to be attached to walls or ceilings). Light types may have to be chosen by discrete optimization from a set of known types, each of which would have many fewer DOFs than the fancy all-in-one light type used in production. Finally, though architects may choose to establish "dramatic" lighting, the rendering needs to be realistic rather than cinematic, favoring GI methods. Thus we would like to further explore methods for accelerating the rendering-optimization loop in this context, perhaps exploiting the aforementioned constraints on light positions as well as strong literature for fast GI.

New Interface Paradigms. It would be interesting to experiment with additional interface paradigms, such as combining our painting approach with sketching and gestural interfaces as well as direct manipulation methods on the effects of lights. Also it would be interesting to provide our painting metaphor to define surface and light characteristics together while painting, for example, allowing the degrees of freedom of the surface shaders to vary along with those of the lights.

Reducing Number of Lights. Figure 7 shows an example where the lighter removes a light while trying to preserve the effect of the omitted light. As mentioned in the introduction, production lighting rigs tend to have many lights. However, there are several potential benefits to reducing this number. First, it makes it easier for the lighter to manage because sorting through tens or hundreds of lights, trying to find an offending one can be a chore. Second, since rendering time scales with the number of lights this idea might be employed as a batch postprocess to lighting for the purpose of accelerating offline render times (which is often hours or even days per-frame). The challenge for this kind of application is that it couples both discrete and continuous optimization as ideally the computer would discover which lights to remove while figuring out how to compensate.

ACKNOWLEDGMENTS

We would like to thank Jean-Claude Kalache for originally motivating this work and for his helpful guidance. We also would like to thank Dani Lischinski, Alex Mohr, Szymon Rusinkiewicz, and John Warren for their suggestions and encouragement about this work. We also thank the Digital Michelangelo Project, Staford University, for the use of the David model; for others models in this article, our thanks to Jeff Budsberg, Jose Guerra Chapa, Sae-Kyoung Lee, Wei Han, Wonju Shin, Jeff Wang, and Nasheet Zaman and members of the Program of Computer Graphics, Cornell University.

REFERENCES

ALTON, J. 1949. *Painting With Light*. Republished in 1995 by University of California Press, Berkeley, CA.

ANRYS, F., DUTRE, P., AND WILLEMS, Y. D. 2004. Image based lighting design. In *the 4th IASTED International Conference on Visualization, Imaging, and Image Processing*.

BARZEL, R. 1997. Lighting controls for computer cinematography. *J. Graph. Tools 2*, 1, 1–20.

COSTA, A. C., SOUSA, A. A., AND FERREIRA, F. N. 1999. Lighting design: A goal based approach using optimisation. In *Eurographics Workshop on Rendering*.

DORSEY, J. O., SILLION, F. X., AND GREENBERG, D. P. 1991. Design and simulation of opera lighting and projection effects. In *Proceedings of SIGGRAPH*. Vol. 25. 41–50.

GERSHBEIN, R. AND HANRAHAN, P. M. 2000. A fast relighting engine for interactive cinematic lighting design. In *Proceedings of ACM SIGGRAPH*. 353–358.

GUMHOLD, S. 2002. Maximum entropy light source placement. In *Proceedings of IEEE Visualization*. 275–282.

HANRAHAN, P. AND HAEBERLI, P. E. 1990. Direct WYSIWYG painting and texturing on 3D shapes. In *Proceedings of SIGGRAPH*. Vol. 24. 215–223.

KALACHE, J.-C. 2004. Personal communication with several lighters at Pixar Animation Studios, most notably Jean-Claude Kalache.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Trans. Graph. 21*, 3 (July), 755–762.

KAWAI, J. K., PAINTER, J. S., AND COHEN, M. F. 1993. Radioptimization—goal based rendering. In *Proceedings of SIGGRAPH*. 147–154.

LOWELL, R. 1992. *Matters of Light & Depth*. Lowel-Light Manufacturing Inc., New York, NY.

MARKS, J., ANDALMAN, B., BEARDSLEY, P. A., FREEMAN, W., GIBSON, S., HODGINS, J. K., KANG, T., MIRTICH, B., PFISTER, H., RUML, W., RYALL, K., SEIMS, J., AND SHIEBER, S. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of SIGGRAPH*. 389–400.

MOHAN, A., TUMBLIN, J., BODENHEIMER, B., GRIMM, C., AND BAILEY, R. 2005. Table-top computed lighting for practical digital photography. In *16th Eurographics Workshop on Rendering*. 165–172.

NELDER, J. AND MEAD, R. 1965. A simplex method for function minimization. *Comput. J. 7*, 308–311.

PATOW, G. AND PUEYO, X. 2003. A survey of inverse rendering problems. *Comput. Graph. For. 22*, 4 (Dec.), 663–687.

PELLACINI, F., TOLE, P., AND GREENBERG, D. P. 2002. A user interface for interactive cinematic shadow design. *ACM Trans. Graph. 21*, 3 (July), 563–566.

PELLACINI, F. AND VIDIMCE, K. 2004. A lighting model for cinematic lighting using CG. In R. Fernando, Ed. *GPU Gems*, Addison-Wesley, Reading, MA.

PELLACINI, F., VIDIMCE, K., LEFOHN, A., MOHR, A., LEONE, M., AND WARREN, J. 2005. Lpics: A hybrid hardware-accelerated relighting engine for computer cinematography. *ACM Trans. Graph. 24*, 3 (Aug.), 464–470.

POULIN, P. AND FOURNIER, A. 1992. Lights from highlights and shadows. In *Symposium on Interactive 3D Graphics*. Vol. 25. 31–38.

POULIN, P. AND FOURNIER, A. 1995. Painting surface characteristics. In *Eurographics Rendering Workshop*. 160–169.

POULIN, P., RATIB, K., AND JACQUES, M. 1997. Sketching shadows and highlights to position lights. In *Computer Graphics International*.

POWELL, M. J. D. 1994. A direct search optimization that models the objective and constraint functions by linear interpolation. In S. Gomez and J. P. Hennart, Eds. *Advances in Optimzation and Numerical Analysis*, Kluwer Academic, Dordrecht, Germany, 51–67.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proceedings of ACM SIGGRAPH*. 579–584.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1995. *Numerical Recipes in C* 2nd Ed. Cambridge University Press.

RASKAR, R. AND COHEN, M. F. 1999. Image precision silhouette edges. In *ACM Symposium on Interactive 3D Graphics*. 135–140.

SCHOENEMAN, C., DORSEY, J., SMITS, B., ARVO, J., AND GREENBERG, D. 1993. Painting with light. In *Proceedings of SIGGRAPH*. 143–146.

SHACKED, R. AND LISCHINSKI, D. 2001. Automatic lighting design using a perceptual quality metric. *Comput. Graph. Foru. 20*, 3, 215–226.

SHIRLEY, P. AND MORLEY, R. 2003. *Realistic Ray Tracing* 2nd Ed. AK Peters.

WANG, Y. AND SAMARAS, D. 2002. Estimation of multiple illuminants from a single image of arbitrary known geometry. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 272–288.

Received April 2006; revised February 2007; accepted April 2007