Eurographics Symposium on Rendering 2008
Steve Marschner and Michael Wimmer
(Guest Editors)

Volume 27 (2008), Number 4

iCheat: A Representation for Artistic Control of Indirect Cinematic Lighting

Juraj Obert[1†] Jaroslav Křivánek[2] Fabio Pellacini[3] Daniel Sýkora[2] Sumanta Pattanaik[1]

[1]University of Central Florida
[2]Czech Technical University in Prague
[3]Dartmouth College

Abstract

Thanks to an increase in rendering efficiency, indirect illumination has recently begun to be integrated in cinematic lighting design, an application where physical accuracy is less important than careful control of scene appearance. This paper presents a comprehensive, efficient, and intuitive representation for artistic control of indirect illumination. We encode user's adjustments to indirect lighting as scale and offset coefficients of the transfer operator. We take advantage of the nature of indirect illumination and of the edits themselves to efficiently sample and compress them. A major benefit of this sampled representation, compared to encoding adjustments as procedural shaders, is the renderer-independence. This allowed us to easily implement several tools to produce our final images: an interactive relighting engine to view adjustments, a painting interface to define them, and a final renderer to render high quality results. We demonstrate edits to scenes with diffuse and glossy surfaces and animation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism — Rendering, Global Illumination

1. Introduction

Lighting plays a crucial role in computer cinematography, where it supports storytelling, enhances mood and directs viewer attention [Alt95]. For this very reason, the goal of cinematic lighting design is not achieving physical realism, but obtaining a carefully controlled illumination, however unphysical that may be. For direct illumination, a long tradition of lighting "cheats" has developed over the years, e.g. arbitrary falloffs, fake blockers [Bar97], and shadows from tweaked positions [PTG02], just to name a few. To fit production shader-based workflow, these adjustments are encoded procedurally in complex light shaders [Bar97, PVL*05].

Indirect illumination did not play an important role in cinematic lighting until recently, when an increase in its efficiency has made it viable for cinematic scenes [TL04, CFLB06]. To this day though, artistic control of indirect illumination has been limited to simple adjustments and has not reached the flexibility common to direct lighting. An example of a common practice is the procedural modification of surface and light shaders to change the color of the emitted indirect light, as well as including and excluding objects from indirect transport [TL04]. While this works well for some adjustments, many effects cannot be expressed in simple manners just by altering shaders, in stark contrast with direct illumination where modifying shader code is comprehensive and efficient. Furthermore, for an intuitive and efficient lighting design, artists seek direct control over the *interaction* between materials, geometry and direct illumination, rather than their independent adjustments.

This paper introduces a *representation* for artistic control of indirect illumination that is *comprehensive*, *efficient*, *intuitive* and not tailored to any software architecture or way of computing global illumination (i.e. *renderer-independent*). The key insight of our method is to encode indirect illumination "cheats" as scale and offset values of the transport between points in the scene, and by keyframing them to support animation. Example edits made using our representation are shown in Figure 1. Rather than focusing on supporting

† e-mail: jobert@cs.ucf.edu

Figure 1: *Artistically modified indirect illumination. The left image shows original unmodified global illumination. In the middle image we directed more indirect light at the character, bringing her into focus. In the right image we modified hue/saturation of indirect lighting hitting the room and changed its directionality to make the character appear lit from her right-hand side.*

specific effects, this representation is general in its support of any adjustment since artists can arbitrarily alter the transport between scene locations. To the best of our knowledge, our work is the first to propose a general formulation that encompasses any user edits in a simple framework.

To make this efficient, we take advantage of the sparse, low-frequency nature of user edits to encode the adjustments efficiently in an approximate form that makes offline rendering practical while supporting interactive lighting design. Furthermore, since artists control transport directly, the representation is intuitive and naturally supports simple user interfaces, such as painting-based metaphors. Finally, our representation does not rely on procedurally altering shader code, which is renderer specific, but is defined by sampling a simple mathematical operator that can be implemented in any renderer and user interface.

The benefit of having a renderer-independent representation has been clear during the development of this project, where we have been able to integrate a real-time relighting engine to preview our edits, a user interface to interactively perform the adjustments by painting, and an offline renderer for final high quality output. Furthermore, to achieve highest efficiency our two renderers used different algorithms for indirect lighting, making it simply impossible to support procedural alterations. This simplicity is in contrast with the complexity of matching light shaders between real-time and offline rendering in today's procedural shader-based pipelines, as demonstrated in the Lpics and Lightspeed systems [PVL*05, RKKS*07].

The remainder of this paper introduces our adjustment representation and demonstrates its benefits in the context of cinematic lighting design. While any number of bounces can be used to compute global illumination, the user only controls the last one, since this was found to be more controllable for artists and sufficient for cinematic purposes [TL04]. In summary, this paper makes the following contributions:

- we introduce a general and intuitive representation for artistic control of indirect illumination
- we demonstrate how arbitrary edits can be encoded in such formulation, including support for animated scenes
- we take advantage of the sparse, low frequency nature of user edits to encode them efficiently
- we derive a real-time rendering algorithm to preview edits, and show a simple painting interface to control them
- we demonstrate the feasibility of inclusion in a production pipeline by rendering all our final edits in an offline renderer that supports indirect illumination in a way similar to Renderman.

2. Related Work

Indirect illumination algorithms can provide remarkable realism when rendering synthetic scenes. Reviewing the variety of methods available is beyond the scope of this paper and we refer to reader to [DBB06] for a recent survey. In most production renderers for cinematic lighting, it is common to use final gathering algorithms to compute irradiance from scene surfaces [CFLB06]. While in most cases only one bounce is considered [TL04, CFLB06], photon mapping is available to artists that require the computation of multiple indirect bounces [Jen01].

In today's productions, two common approaches are used to artistically control indirect illumination. First, by rendering the scene in layers, adjustments to final colors can be performed with compositing tools. While these methods allow for global adjustment to be efficiently performed, adjustments that only affect parts of the scenes remain too cumbersome with these methods. Our method supports any adjustment, whether local or global, in an intuitive and efficient manner. The practice most closely related to our own is the idea of controlling the computation of indirect illumination directly by procedurally altering shader code, exemplified in the production work in today's computer generated movies [TL04, CFLB06]. Compared to this approach our method has significant benefits in generality, intuitiveness and practicality that we have already discussed in the introduction and we will further review in the following sections.

One of the advantages of our representation is that it can be easily supported in interactive relighting engines that simplify lighting design by interactively displaying the adjusted illumination during editing [PVL*05, RKKS*07]. Of the various algorithms available, we have implemented a variation of [HPB06] since this algorithm was designed exactly to support indirect illumination for cinematic scenes and since our representation maps well to their relighting framework. Section 5 will discuss our modifications.

While our representation does not impose a specific user interface, one of its advantages is that is can support an intuitive painting metaphor to directly control indirect effects. We used such a user interface to generate the edits in this article and noticed that controlling complex adjustments is quite simple and efficient. The idea of using a painting metaphor is loosely related to goal-based lighting design, where artists specify how the final scene should look, often via painting, while an algorithm automatically sets scene parameters to best fit artists requirements [KPC93, SDS*93, OMIS06, PBMF07]. We departed from this paradigm in our prototype since estimating parameters via optimization is quite slow in the case of indirect illumination and may not converge to a reasonable solution. The reader should notice, though, that our representation allows artists to choose whichever user interface they feel most comfortable with.

3. Representation

Our goal is to artistically adjust the indirect lighting for each frame of a cinematic sequence. To allow for maximum control, we specify these adjustments independently for each frame in the sequence. In the following paragraphs, we describe our method focusing on a single frame, and present its extensions to animated scenes later.

For the fixed camera position of the rendered frame, the indirect illumination B of a *view* point v can be written as

$$\begin{aligned} B(v) &= \int_{g\in S} B(g)T(v,g)dg = \\ &= \int_{g\in S} B(g)\left(\rho(v,g)G(v,g)V(v,g)\right)dg \end{aligned} \tag{1}$$

where g is a *gather* point, S the set of all positions on object surfaces, $T(v,g)$ is the transport operator, $\rho(v,g)$ is the BRDF evaluated between v and g, $G(v,g) = -(N_v \cdot \overrightarrow{vg})(N_g \cdot \overrightarrow{vg})/|\overrightarrow{vg}|^2$ is the geometric term and V the visibility. Note that while ρ can be glossy in the last bounce, we assume that indirect effects only depend on the diffuse radiance of all other bounces as is common in cinematic lighting [TL04]. Current methods to control indirect illumination encode adjustments as procedural modifications of the various terms ρ, G, V, $B(g)$ [TL04]. While many effects can be represented this way, adjustments that depend on the interaction between these terms are cumbersome to capture and very hard to control for artists. Furthermore, since most of these adjustments are represented as shaders, it is often close to impossible to have multiple renderers compute the same image.

3.1. Edit Representation

We propose a representation where artists control indirect illumination by directly altering the transport coefficients $T(v,g)$. Formally, we define the edited transport T' by scaling and offsetting the original values

$$T'(v,g) = s(v,g)T(v,g) + o(v,g) \tag{2}$$

where s and o are scale and offset functions defined over the cartesian product of the scene locations $S \times S$. We support color adjustments by storing scale/offsets for each channel separately. This formulation can *represent any edit* an artist might want to achieve since any value of the transport coefficient can be obtained; we demonstrate several examples in the next section. Furthermore, users can control the values of the adjustments s and o directly and intuitively, without having to manipulate the individual components separately.

We sample the functions s and o over a set of view and gather samples v_i and g_j. In our implementation, we choose n view samples as the points visible through all pixels, and m gather samples uniformly distributed on the scene geometry as in [HPB06]. In this formulation, the function s and o can be thought of as $n \times m$ scale and offset matrices $[S_{ij}]$ and $[O_{ij}]$

$$s(v,g) \approx [S_{ij}] \qquad o(v,g) \approx [O_{ij}] \tag{3}$$

We interpolate the matrix values to reconstruct scales and offsets for arbitrary view/gather pairs, as described in Section 5.

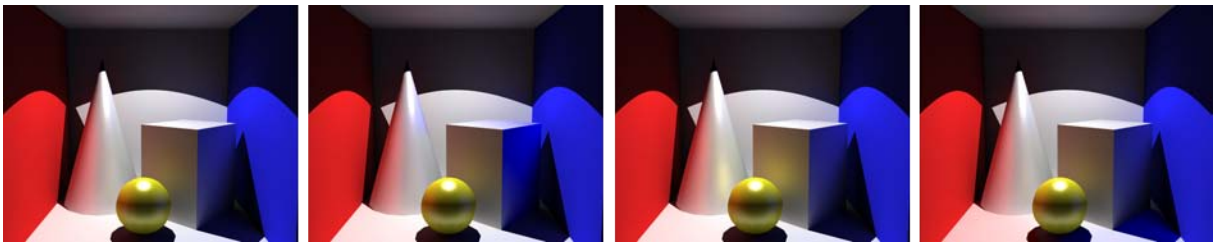
3.2. Efficient Encoding

High sampling density of scale and offset matrices is required to achieve high quality results. For our results, we use 360K view samples and 64K gather samples, making the direct storage of the matrices impractical. To make the storage tractable, we project each row of the matrices in 1D Haar wavelets and cull many of the less important wavelet coefficients

$$S^w \approx WS \qquad O^w \approx WO \tag{4}$$

where W is the wavelet transform matrix. Similarly to prior work on cinematic relighting [HPB06], we impose the wavelet basis by hierarchical clustering of the gather points to ensure spatial coherence after wavelet projection. In our experience, wavelets capture well the all-frequency structure of user edits. In our prototype we use 100 wavelet coefficients per row.

Furthermore, many edits can be represented as constant rows or columns of the matrices. These can be equivalently expressed as adjustments to $B(v_i)$ and $B(g_j)$ respectively. To take direct advantage of this, we additionally store scale and



(a) Original (b) 4D Hue/Sat/Value (c) 4D Quadratic falloff coefficient (d) 4D Painting selection

Figure 2: *Different 4D edits were applied to original indirect illumination in image (a). We modified hue and saturation of indirect lighting reflected off the blue wall towards the glossy cone and the diffuse box in image (b). In image (c), we decreased the quadratic falloff coefficient from the glossy sphere towards the cone and the box. And finally, in image (d) we painted a 4D selection to emphasize color bleeding on the floor.*

offsets vectors $\{s_i^v, o_i^v\}$ and $\{s_j^g, o_j^g\}$ respectively for view and gather samples to obtain

$$B'(v_i) = s_i^v B(v_i) + o_i^v \qquad B'(g_i) = s_i^g B(g_i) + o_i^g \quad (5)$$

3.3. Discussion

The reader may wonder why we represent edits as scale and offset of the transport coefficients T, rather than simply storing the edited transport coefficients T'. In this respect, our representation has three main advantages. First, storing the transfer coefficients directly for a high quality offline rendering would be impractical, while user edits are significantly sparser and lower frequency. Second, offline and real-time rendering algorithms have different quality/efficiency tradeoffs when computing indirect illumination; our sampled representation can be integrated with these algorithms while preserving such tradeoffs. Third, and possibly most importantly, it is more intuitive to modify scale/offsets than the transport itself; for example to darken a wall, users can just scale the radiance by a constant, while still maintaining these beautiful realistic gradients so hard to achieve without indirect illumination.

3.4. Animation

We support animation assuming artists follow the same workflow typical of direct illumination, where adjustments are defined for a few keyframes in the sequence and interpolated in the remaining frames (see Figure 4). In our framework, we interpolate matrices and vectors between keyframes. This requires determining correspondence between matrix elements. We make this correspondence implicit in gather samples by choosing the same *parametric* surface location at each frame. View sample sets from the keyframes are reprojected to the current frame, too. We compute a matrix row for an arbitrary view point v in the current frame by performing spatial interpolation for two surrounding keyframes using their reprojected view sample positions (we take location and normal orientation into account when performing the lookup), and then temporally interpolate the two resulting rows. Thus, if a large area that was occluded in the key frame becomes visible in the in-between frame, artifacts may, indeed, appear.

4. Example Edits and Workflow

This section demonstrates the generality of our representation by performing several edits to a few scenes lit with indirect illumination. To generate all results in this paper, we design the edits in an interactive relighting engine that supports our representation and then export them to an offline renderer for high quality imagery. This workflow takes direct advantage of the fact that our representation is renderer-independent and can be easily supported using different algorithms in any renderer, just like indirect illumination is. Furthermore, the use of an interactive tool for previewing lighting coupled with a high quality offline rendering for final output is typical of current cinematic lighting workflow [PVL*05, RKKS*07], further proving the practicality of our representation. We present implementation details in the following section.

We control indirect illumination by first selecting view and gather samples (4D selection), using a painting interface, then applying arbitrary operations to the transport coefficients which are finally mapped to scale and offset values. While our paper does not focus on a specific workflow, we found this process intuitive and fast and thus we will present it in this chapter as an example of possible integration with cinematic production tools. Also, we remind the readers that the edits shown are just examples of what our representation can achieve, but we expect different artists and cinematic "styles" to require a variety of different effects. Our framework naturally supports any new effect, and has the substantial benefit that new edits can be implemented simply by altering the scale/offset coefficients and do not require any other change in the renderer and production assets (geometry, shaders, etc.).

4.1. Example Workflow

Indirect edits start by selecting view and gather samples using a painting interface. This essentially selects the block of the transport matrix at the intersection between the rows and columns corresponding to the selected view and gather samples respectively. Our prototype provides both painting

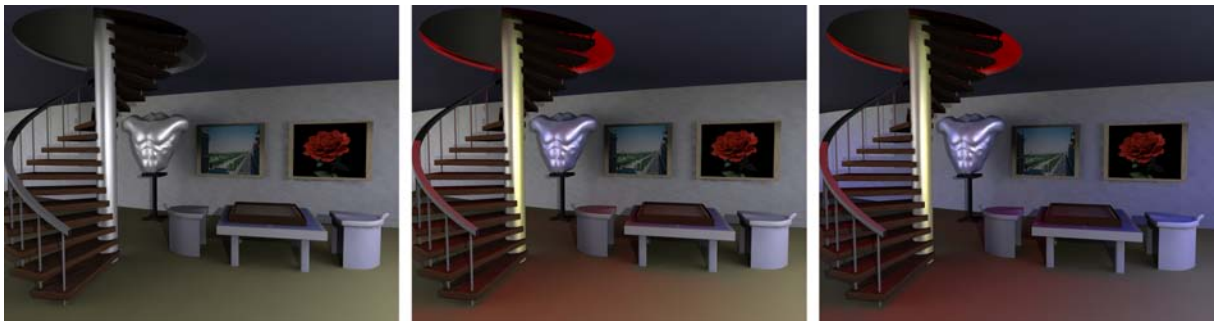


Figure 3: *Examples of edits on glossy surfaces. The left image shows original rendering with global illumination and the remaining two images show our edits. In the middle image we made the cylinder above the staircase emit red indirect light, changed indirect hue on the glossy staircase piece and on the glossy statue torso. In the right image, we added a more bluish tone by reflecting more indirect light off the ceiling towards the floor and the wall.*

and per-object selection mechanisms and supports smooth, non-binary selections. Arbitrary modifications are then applied to this block by projecting user edits to scale and offset coefficients and compressing them in the wavelet domain.

In our examples, we fixed the main camera for simplicity of explanation. However, our example workflow is more similar to a lighting design system in production. In the real-time renderer, the main camera is fixed (in a way similar to [PVL*05]) and other viewports are provided for the user to navigate around the scene and work with surfaces not visible from the main camera.

4.2. Example edits

Figure 2 shows three samples edits, where we manipulate hue/saturation/value of reflected indirect light in (b), quadratic falloff coefficient in (c) and directly paint additive coefficients in (d). In the first example, we selected the blue wall as the casting object, the box and the cone as receiving objects and increased saturation. Notice that glossy objects are handled as easily as diffuse in this representation (see Figure 3 for more examples of glossy edits). In the second example, we changed the quadratic falloff coefficient of light reflected off the sphere and thus made more light arrive at the box and the cone. The third example is similar to the first, but uses painted selection (the floor) instead of per-object selection. Note that these are just a few example of possible edits. We also have successfully experimented with removing indirect shadows, or selectively increasing object albedos for indirect lighting, etc.

5. Implementation

We have implemented our representation in two applications: an interactive relighting engine to preview the edits coupled with a painting interface to perform them, and an offline renderer to compute final frames. For each, we have adapted a known algorithm to support our editing representation. All results in this paper were created by concurrent use of these two tools. Having a consistent representation for lighting cheats made it possible to have them interact efficiently. This section describes implementation details of the interactive relighting engine and the offline renderer. Once again, we remind the reader that these are just examples of how easy it is to adapt rendering algorithms to support our representation; we expect others to easily include our edits in their preferred systems.

5.1. Interactive Relighting Engine

Our interactive relighting algorithm is based on the direct-to-indirect transfer approach of [HPB06]. In their algorithm a transport matrix T is precomputed between a set of view samples v_i, chosen as the pixel centers, and a set of gather samples g_j, uniformly sampled in the scene. The precomputed matrix is compressed by projecting each row in wavelet space and culling small coefficients to obtain an approximate sparse matrix T^w. For each change in the lighting, the direct illumination of gather samples is computed and multiplied column-wise by the sparse transfer matrix. In our implementation, we also encode each row sparsely, but use row-wise instead of column-wise multiplication. To support indirect illumination cheats, scale and offset matrices are projected in the same wavelet basis as the transport operator and lossily compressed. To support animation, we simply keep in memory the transfer matrix and the scale/offset coefficients for each keyframe.

5.2. Edit Updates

As described above, the interactive renderer is unaware of how scale and offset coefficients are modified. During an editing session, it is the responsibility of the user interface code to update these coefficients interactively, as would normally happen with direct illumination light parameters. Once again, our simple representation makes this decoupling possible. We compute scale/offset coefficients every time the user commits an edit, every time direct lighting changes and every time a non-linear edit is applied (such as gamma edit, which is non-linear with respect to lighting values).

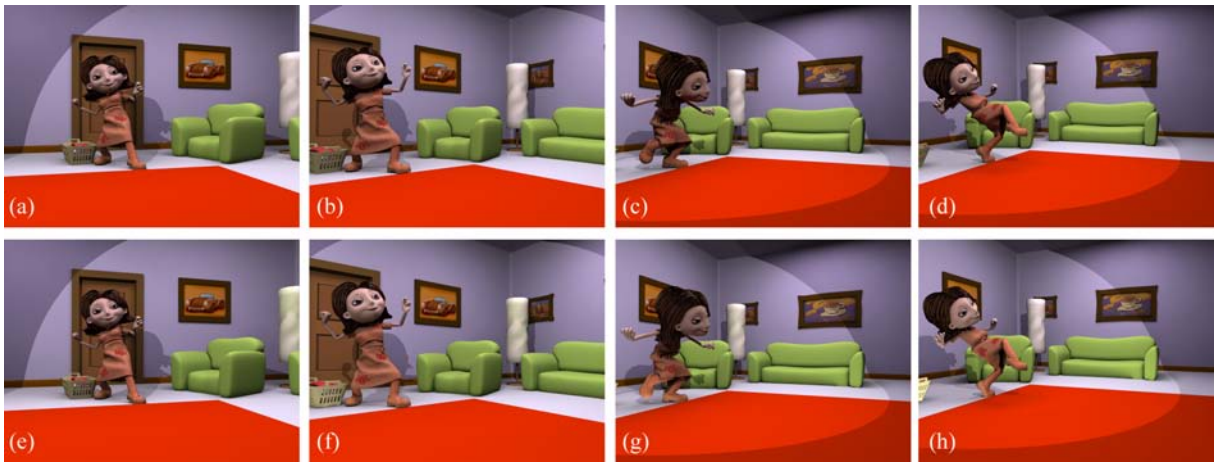


Figure 4: *Keyframed edits. Original (first row) and modified (second row) frames from an animated sequence are shown. Frames (e) and (h) were selected as keyframes and lighting was designed for them. Frame (e) was left intact and in frame (h), indirect lighting arriving at the character, laundry basket and the couch was modified. Edits in images (f) and (g) were interpolated and show smooth transition of the edits between two keyframes.*

Sample edits in Figure 2 are implemented as follows. In Figure 2(c) we show editing of the quadratic falloff coefficient. This is an example of a geometry edit, i.e. an edit that alters the geometry term of the rendering equation. Our implementation of this edit consists of unrolling the matrix row from the wavelet space, then multiplication of matrix elements by a constant and finally projection back into wavelets. Then we divide the original and new matrix elements to obtain scales and offsets.

Using a row-based encoding lets us access the matrix quickly since we only need rows corresponding to edited view samples. To gain further speed, we adaptively subsample the rows using irradiance caching [WRC88]. Wavelet triple product [NRH04] can further optimize the updates and is left for future work.

Figure 2(b) demonstrates editing of hue/saturation/value of indirect lighting computed between two sets of points. This is a non-geometry edit, as it doesn't require modification of the geometry term. In our implementation we first compute light transport between two sets of points using the underlying rendering algorithm [HPB06]. We store this information as a 2D texture (`tex1`). Then, we execute the user edits (i.e. change HSV in this example) and store the result as another texture (`tex2`). To obtain scales and offsets, we divide `tex2` by `tex1`. If division by zero occurs, offsets are set to values from `tex2` and scales are set to zero. Otherwise, scales are set to the result of the division and offsets are set to zero. Implementation of the edits in Figure 2(d) is the same, the images only differ in the selection mechanism used (per-object vs. painting).

It is important to realize that even though multiple edits can be applied over the same sets of gather/view points, the wavelet compression doesn't cause any major loss in quality for two reasons. First, we keep edits uncompressed while the user works with them and only project them into wavelets when they have been committed (i.e. finalized). Second, we only store modifications to the matrix, which are inherently of very low frequencies and therefore far less susceptible to cause artifacts.

5.3. Offline Renderer

We compute high quality images using an offline renderer based on final gathering accelerated by irradiance and radiance caching [WRC88,KGPB05]. Multiple bounces are handled by using photon mapping. Our algorithm for indirect lighting is the same as used in most Renderman implementations, showing the feasibility of a deployment in production. While our realtime implementation uses point sampling, we interpolate the matrix rows for our final rendering. For primary rays hitting a surface at view point v, we locate the three nearest view samples (we use a kd-tree to locate nearest view and gather samples) and interpolate the wavelet coefficients in the corresponding matrix rows. The interpolated coefficients are then decoded from the wavelet domain and retained for fast lookup in final gathering.

For each gather point, g, hit by a gather ray emitted from v, we let the renderer compute illumination as usual (e.g. by a photon map lookup or by evaluating direct illumination at g) and subsequently alter the computed illumination by the offset and scale factor looked up from the currently decoded matrix row. To index into the row, we simply locate the nearest gather sample to g. Nearest neighbor filtering is sufficient here, since the illumination is averaged in final gathering. Irradiance caching helps reduce the overhead since the matrix row interpolation and wavelet decoding is only performed when a new cache record is added. Animation is performed temporal edit interpolation as described in Section 3.

Scene	Edits size	T_{iCheat}	T_{normal}
Fig. 1	7MB	4:20	4:00
Fig. 2	7MB	0:55	0:40
Fig. 3	7MB / 9MB	2:40	2:30

Figure 5: *Size of the edits matrices and final renderer performance (in minutes:seconds) for the example scenes in the paper. T_{iCheat} gives the render time with the edits applied, T_{normal} without.*

6. Results

The performance results reported in this section were measured on a PC with Intel Core Duo 6850 processor, 4GB RAM, GeForce 8800 Ultra GPU running Windows XP. We rendered all our images at 640×480 resolution, with up to 36 samples per pixel in the high quality renderer.

The real-time renderer performs at frame rates around 150 FPS for 300k view samples, 64k gather samples, and image resolution of 640×480. The matrix updates are performed at 10 frames per second, a speed still largely sufficient for comfortable work.

The size of edit representation depends on the number of affected view samples and the number of coefficients stored per matrix row (which in turn depends on the nature of the edit). The maximum of 100 coefficients per row was sufficient for all the example edits shown in the paper. Figure 5 summarizes the matrix sizes for various scenes in this paper.

Overhead implied by applying edits in final rendering is, for the most part, due to interpolating and decoding matrix rows and locating the nearest gather sample for each final gather ray. As such, it is proportional to the size of the edit matrices. In our tests, the overhead never exceeded one minute for a frame of 640×480 pixels. The rendering times are summarized in Figure 5.

7. Conclusion and Future Work

We have presented a representation for artistic control of indirect lighting, with application to computer cinematography. Rather than representing edits as procedural shader modifications, we numerically sample the edits using scale and offset coefficients and take advantage of their nature to efficiently compress and render the adjustments. This representation is general, easy to manipulate, and renderer-independent, allowing simple and effective workflow in production environments.

In the future, we are interested in investigating representations of this nature to support adjustments of other aspects of cinematic appearance design, such as materials, subsurface scattering, volumetric effects and tone manipulations.

Acknowledgments

This work has been supported by University of Central Florida I2Lab Fellowship, Florida High-Tech Corridor Research Funding and Ministry of Education Youth and Sports of the Czech Republic under the research program LC-06008, "Center for Computer Graphics". Fabio Pellacini was partly supported by NSF (CNS-070820, CCF-0746117). Many thanks to Vlasta Havran for the Golem ray tracer and to Universal Production Partners (UPP) for the Orion scene.

References

[Alt95] ALTON J.: *Painting with Light*. University of California Press, 1995.

[Bar97] BARZEL R.: Lighting controls for computer cinematography. *Journal of Graphics Tools 2*, 1 (1997), 1–20.

[CFLB06] CHRISTENSEN P., FONG J., LAUR D., BATALI D.: Ray tracing for the movie cars. In *Proc. of IEEE Symposium on Interactive Ray Tracing* (2006), pp. 1–6.

[DBB06] DUTRÉ P., BALA K., BEKAERT P.: *Advanced Global Illumination*, 2nd ed. A K Peters Ltd., 2006.

[HPB06] HAŠAN M., PELLACINI F., BALA K.: Direct-to-indirect transfer for cinematic relighting. *ACM Trans. Graph. (Proc. SIGGRAPH) 25*, 3 (2006), 1089–1097.

[Jen01] JENSEN H. W.: *Realistic Image Synthesis Using Photon Mapping*. A K Peters Ltd., 2001.

[KGPB05] KŘIVÁNEK J., GAUTRON P., PATTANAIK S., BOUATOUCH K.: Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (September/October 2005).

[KPC93] KAWAI J. K., PAINTER J. S., COHEN M. F.: Radioptimization: Goal based rendering. In *SIGGRAPH'93 Proceedings* (1993), pp. 147–154.

[NRH04] NG R., RAMAMOORTHI R., HANRAHAN P.: Triple product wavelet integrals for all-frequency relighting. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3 (2004), 477–487.

[OMIS06] OKABE M., MATSUSHITA Y., IGARASHI T., SHUM H.-Y.: *Illumination Brush: Interactive Design of Image-based Lighting*. Tech. Rep. MSR-TR-2006-112, Microsoft Research, 2006.

[PBMF07] PELLACINI F., BATTAGLIA F., MORLEY R. K., FINKELSTEIN A.: Lighting with paint. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 2 (2007).

[PTG02] PELLACINI F., TOLE P., GREENBERG D. P.: A user interface for interactive cinematic shadow design. *ACM Trans. Graph. (Proc. SIGGRAPH) 21*, 3 (2002), 563–566.

[PVL*05] PELLACINI F., VIDIMČE K., LEFOHN A., MOHR A., LEONE M., WARREN J.: Lpics: A hybrid hardware-accelerated relighting engine for computer cinematography. *ACM Trans. Graph. (Proc. SIGGRAPH) 24*, 3 (2005), 464–470.

[RKKS*07] RAGAN-KELLEY J., KILPATRICK C., SMITH B. W., EPPS D., GREEN P., HERY C., DURAND F.: The light-speed automatic interactive lighting preview system. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 3 (2007).

[SDS*93] SCHOENEMAN C., DORSEY J., SMITS B., ARVO J., GREENBERG D.: Painting with light. In *SIGGRAPH'93 Proceedings* (1993), pp. 143–146.

[TL04] TABELLION E., LAMORLETTE A.: An approximate global illumination system for computer generated films. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3 (2004), 469–476.

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *SIGGRAPH'88 Proceedings* (1988), pp. 85–92.