

# Efficient rendering of light and camera animation for navigating a frame array

Alex Méndez-Feliu  
Universitat de Girona  
amendez@ima.udg.es  
<http://ima.udg.es/~amendez/>

Mateu Sbert  
Universitat de Girona  
mateu@ima.udg.es  
<http://ima.udg.es/~mateu/>

## Abstract

Computing a series of frames of an animation in a global illumination framework is a very costly process. If we want in addition to experiment with different camera and lighting positions, the cost can become prohibitive. In this paper we show how this computation time can be dramatically reduced by combining different simplifications and the reuse of previously computed data. On the one hand, the data between neighbor frames in a walkthrough (camera animation) can be reused by reprojection. On the other hand, using the obscurances technique, we simplify the computation of the global illumination by separating the direct and indirect lighting and by darkening the more occluded parts of the objects of the scene. Color bleeding is also applied. Due to the separation of direct and indirect illumination, for a given camera position, the indirect illumination needs to be computed only once, while the direct lighting can be computed for as many different light positions as we like. Both camera and light animations can be efficiently combined giving as result a set of images through which we can navigate and, for example, see the light animation for different camera positions, do the same walkthrough for different locations of the light, or combine both light and camera animations in the same movie.

**Keywords:** Computer Graphics, Three-Dimensional Graphics and realism, Raytracing, Rendering, Animation

## 1 Introduction

In production rendering of most short and long animated films and virtual scenarios of a movie the maximum realism has to be achieved. Nowadays, the most recent global illumination techniques do the work, though they require a big effort in computation power and time. Global illumination aims to simulate the lighting of a scene in a physically realistic way using ray casting based techniques, as *path tracing* [1] and *photon mapping* [2]. In each frame millions of rays have to be cast to allow for an image without disturbing noise and without noticeable flickering between frames.

In this paper we will concentrate on two kinds of acceleration for global illumination: the reuse of information between contiguous frames in a walkthrough (camera animation) and the use of the obscurances technique, actually not a global illumination technique, but a powerful simplification to compute indirect illumination that leads to very good visual results and allows to compute very efficiently the animation of light.

Havran et al. in [3] presented an architecture for reusing information between frames based on reprojecting to neighbor eye positions the hit of the scene given by a ray cast from the eye through one pixel. In [4], Mendez et al. demonstrate that Havran et al.'s technique is biased for general BRDF's and propose an unbiased solution. They also showed that for diffuse BRDF's, both theirs and Havran et al.'s solution are the same. We use this last one, an unweighted av-

erage of the results, for the reuse of obscurances computation between contiguous frames.

Obscurances [5, 6] is an ambient term technique where occlusions are taken into account, although only a local environment is queried. The amount of occlusion will determinate the diffuse effects. A simplified version of obscurances is used in well known commercial renderers such as *Pixar's Photorealistic RenderMan* under the name of *ambient occlusion* [7]. Obscurances, although it is not a global illumination technique, is able to obtain at a low cost high quality realistic looking images that simulate a global illumination solution. In our case, direct illumination and glossy effects are computed with the usual ray-tracing methods ([8]), and diffuse effects are computed using the obscurances techniques.

In [9] obscurances are applied to rendering of animation frames with moving light sources, taking advantage of one of the properties of the obscurances, the decoupling of direct and indirect illumination. Thus, Mendez et al. just needed to compute obscurances once in the first frame and reuse them in the rest of the animation. A different ambient light intensity is estimated for every frame, resulting in beautiful color-changing effects.

In this paper we make one step ahead, combining the reuse of frames in a camera animation with the animation of light sources. The result is a matrix of images which we can navigate to see the movement of the light from different fixed camera positions, a walkthrough with different light positions, or combine both movements in an animation.

This paper is organized in the following way. First we will introduce briefly the obscurances concept. Next, we will see how the image information can be reused from one eye to its neighbor eye positions. Then, in the next section, we will see the animation of the light for a fixed camera position. The combination of both camera and light animations in one algorithm is shown next. Finally we will present results and the conclusions.

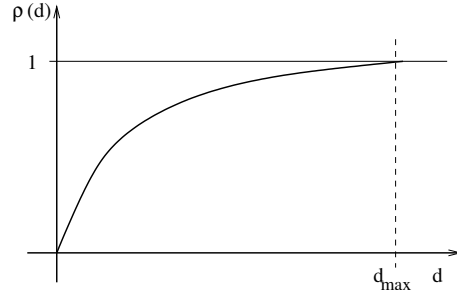


Figure 1: Shape of  $\rho(d)$  function.

## 2 Obscurances

In [5, 6] the obscurances illumination model was defined. Obscurances take account of secondary diffuse illumination, being totally decoupled from direct illumination. For every point in the scene an obscurance value can be computed by scrutinizing the space around it and measuring how occluded is the point with respect to the rest of the scene. In [10] color bleeding is introduced.

The obscurances equation is, then, as follows:

$$W(P) = \frac{1}{\pi} \int_{\omega \in \Omega} R(Q) \rho(d(P, \omega)) \cos \theta d\omega \quad (1)$$

where

- $\rho(d(P, \omega))$ : function with values between 0 and 1, and giving the magnitude of ambient light incoming from direction  $\omega$ . The shape of this function is shown in figure 1.
- $d(P, \omega)$ : distance from  $P$  to the first intersected point in direction  $\omega$
- $\theta$ : angle between direction  $\omega$  and the normal at  $P$
- $R(Q)$ : Reflectivity at the point  $Q$  seen from  $P$  in direction  $\omega$ , for color bleeding purposes.
- $1/\pi$  is the normalization factor.

The indirect illumination for point  $P$  is found multiplying the obscurances factor  $W(P)$  to the reflectivity at the point ( $R(P)$ ) and to an average ambient intensity of the scene ( $I_A$ ):

$$I(P) = R(P) \times I_A \times W(P) \quad (2)$$

Direct illumination and the specular contribution, if needed, are added to (2) to obtain the final illumination at the point. The direct lighting is obtained by sending shadow rays to the light sources. The specular contribution is obtained in a recursive way, by computing all illumination, including again direct, indirect (with obscurances) and specular illumination, if needed, at the reflected point.

### 3 Reuse of frames

We review in this section related work on camera and light animation.

#### 3.1 Camera animation

In [3], an architecture to reuse the information computed for one frame to other frames is presented. From the eye, a ray is cast through a pixel to find a hit point in the scene (named *native hit* to distinguish it from the *outer hits* found through other eyes) and a sample of the radiant flux to the eye is taken using any suitable technique as path tracing or bidirectional path tracing. If the point is from a diffuse surface, the radiance information for that point can be reused in the surrounding frames by reprojecting the point (supposing visibility) and finding the corresponding pixel, where the sampled values are simply averaged. Fig. 2 shows the idea. In [4] the same idea is discussed under the point of view of path reuse using multiple importance sampling (see [11]), as different generation probabilities are involved for non-diffuse BRDF's with different points of view. A new algorithm is presented then to deal with non-diffuse materials, but it requires big amount of memory as all radiances, probabilities, and other data to be combined have to be saved until the final computation of the image.

Now suppose we want to compute a series of frames in a camera animation using ray-tracing with obscurances, as seen in previous section and explained in [9], instead of a true global illumination technique. In this case the direct lighting of a point in a diffuse surface (or the diffuse part of its BRDF, as we divide the BRDF in its pure diffuse and pure specular part) and the obscurances for indirect lighting can be reused in

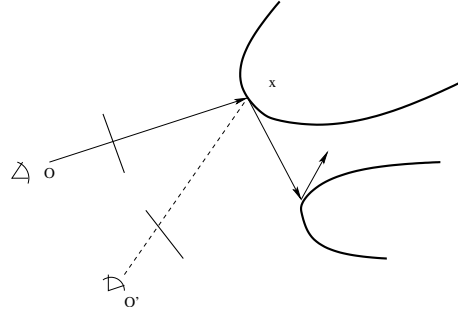


Figure 2: Reusing the path from observer  $O$  for observer  $O'$ , at the cost of the visibility test  $vis(O', x)$ .

multiple frames using Havran et al's technique. To deal with the specular part of the BRDF's we don't have the choice of reusing it as the probability of following the path starting from another eye is zero in the case of pure specular materials. In consequence, the pixels that show specular objects will need more samples or they will have less quality than the diffuse ones.

#### 3.2 Light animation

In recent papers as [12, 13], path-reuse has been applied to light source animation. In [9], the indirect lighting computed with obscurances is reused between frames. Instead of computing every frame from scratch, they take advantage of the fact that neither the obscurances nor the hit points have to be recomputed. This is possible if the camera and all objects in the scene are fixed, and only lights move from frame to frame. Thus they compute first the obscurances and store the hit points and incoming directions of the eye rays. Then, for each frame, direct illumination and specular effects are computed using shadow and specular rays using the stored hit points and directions. This has the side effect of increasing frame-to-frame coherence, eliminating flickering.

To compute indirect light, ambient light factor can be estimated independently for every frame. Thus, only direct light, specular effects, and an ambient light factor, have to be recomputed when light sources move or change intensity with respect to the next frame.

Average light intensity and average reflectivity can be accurately estimated by shooting a few hundred rays from the light sources before-

hand, and follow their paths while computing statistically the parameters needed. In this way ambient intensity, average reflectivity and the area reached by the light are easily computed. Ambient intensity is computed for every frame, and used to multiply the obscurances factor and diffuse color to obtain indirect lighting. If light sources power changes among frames or light movement causes changes in occlusion conditions, ambient intensity also changes and result in beautiful effects in the animation.

#### 4 Combination of camera and light animation

Until now, we have seen on the one hand how to reuse information for neighbor frames in a camera animation and on the other hand how to animate lights reusing the indirect lighting from a fixed camera position. From now on we will combine both solutions in a unique algorithm.

The main idea is to compute simultaneously all combinations of camera and light animation, this is, for every eye position we will compute the images for all light positions. This is represented in figure 3. Of course this results in a high number of images but if we want to generate movies with different combinations of the light and camera animations, we clearly save time.

We keep in memory only the images to reuse, this is, for the current eye position, the indirect light image and an array of  $N_l$  direct and specular images, being  $N_l$  the number of light positions. For every neighbor eye in which to reuse the current results, we keep the indirect light image and the array of direct light images.

The algorithm (see Alg. 1) goes as follows. As part of the initialization process we compute the average light intensity of the scene for every light position. Then, for every pixel in every frame in the camera animation we get as many hits in the scene as samples we need for pixel. Once we have a hit we compute the obscurance value for the indirect color, the direct lighting and the specular values if the BRDF of the object has specular component. These specular and direct values have to be computed for every light position. The specular values are the ones that add more computation time to our algorithm, as a new ray is sent in the reflection

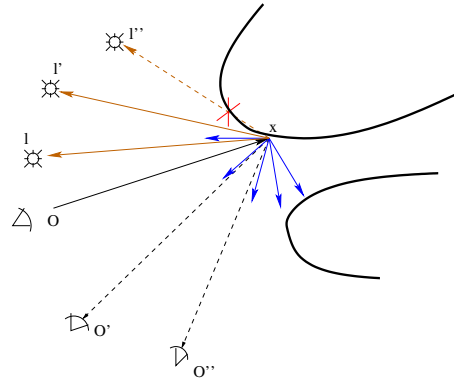


Figure 3: The eyes ( $O$ ,  $O'$  and  $O''$ ) are the different positions of the animation of the camera. The suns ( $l$ ,  $l'$  and  $l''$ ) represent the different positions of the light animation. The obscurances value for the hit point  $x$  is computed with the blue rays, and it can be reused for the different camera positions as well as for the different light positions.

direction, a new hit is found and its illumination computed in a recursive way. But the direct lighting values are very fast and easy to compute, specially when we have a point light or a directional light, as we have only to compute visibility in one direction for each light position.

The specular values can not be reused, so we keep them directly as final values. We reuse the values for the different direct lighting positions and the obscurances values in the  $N_r$  previous and the  $N_r$  subsequent frames, as the neighbor eyes are the ones more likely to have visibility with the hit. As they are diffuse color values, they can be directly averaged with the previous results in the corresponding image.

When we have treated all pixels in the current frame, and before we keep treating the ones of the next frame, we have one frame for which we will not reuse any more values, so we can compute its final values, i. e. multiply the indirect color image and the corresponding average intensities and add them to the respective direct and specular images of the array. Then save all images with different direct light positions to disc. At the end of the computation, all remaining images have to be finally computed and saved to disc the same way.



of 70 obscurances samples and 14 direct light samples (per each light position) per pixel. The resolution for all images is  $400 \times 300$  pixels.

The first movie shows the compound animation of a camera and a point light in the cabin of an aircraft. The camera animation by itself would take eight seconds (this is, 196 frames) and the light moves for 2 seconds (48 frames). The final movie takes 10 seconds and we can appreciate that in the first 2 seconds the camera is fixed and the light is moving. The next 2 seconds the camera moves and the light stays still. The last 6 seconds both camera and light are animated. Of course in this case we don't use all 9216 ( $196 \times 48$ ) images generated, but it's worth having the freedom to generate any other movie involving these two animations. Figure 5 shows part of the possibilities we have. They took almost 18 hours to compute, an image every 7 seconds. A single image of the aircraft model without any reuse and only one light position, using 70 rays per pixel for obscurances computation and 14 rays per pixel for direct light computation, takes 260 seconds, more or less, depending on the camera position and the number of pixels that show specular objects.

The second movie shows the animations of the camera moving through a kitchen (12 seconds, 288 frames) combined with an animation of a directional light coming from the windows and changing its angle for two seconds (48 frames). Same as before, it shows only a part of the possible combinations for all 13824 images generated ( $288 \times 48$ , that took almost 32 hours to compute, an image every 8.2 seconds). Note the changes in the average ambient color and intensity. A single image of the kitchen model without any reuse and only one light position, using 70 rays per pixel for the obscurances computation and 14 rays per pixel for direct light computation, takes around 320 seconds.

The third movie is an animation of a camera traveling over the stairs of a theater (6 seconds, 144 frames) combined with an animation (2 seconds, 48 frames) of the angle of a directional light from the seven windows. It also shows the changes in ambient intensity. All 6912 images ( $144 \times 48$ ) took a bit more than 14.5 hours, 7.6 seconds per image. A single image of the stairs model without any reuse and only one light position, using 70 rays per pixel for the obscurances

computation and 14 rays per pixel for direct light computation, takes around 230 seconds.

## 6 Conclusions

In this paper we have seen an algorithm that combines two previous transversal techniques that reuse information between frames in an animation. On the one hand, the camera animation reuses the diffuse illumination of a hit between neighbor frames by reprojection, and on the other hand, using the obscurances technique to simulate global illumination direct and indirect illumination can be decoupled, and the animation of light can be computed reusing the indirect lighting information. Besides the advantage of joining both techniques, our algorithm is capable to generate images for all combinations of both animations, giving freedom to generate different movies from the same set of images. Considering the time of computation per image, the acceleration obtained with our algorithm is lineal with the number of light positions, with a multiplicative constant which depends on the relative cost of indirect and specular lighting. In our experiments with several scenes and 48 light positions the speed-up was between 35 and 40 relative to computing every frame from scratch.

## Acknowledgments

This project has been funded in part with Gametools project from the European VIth Framework and with grant number TIN2004-07451-C03-01 from the Spanish Government.

The kitchen, aircraft and stairs models are courtesy of LightWork Design Ltd. Thanks to professor László Neumann for his colorful ideas.

## References

- [1] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, December 1997. Available from [http://graphics.stanford.edu/papers/veach\\_thesis](http://graphics.stanford.edu/papers/veach_thesis).

- [2] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Natick, MA, 2001.
- [3] Vlastimil Havran, Cyrille Domez, Karol Myszkowski, and Hans-Peter Seidel. An efficient spatio-temporal architecture for animation rendering. In *Proceedings of Eurographics Symposium on Rendering 2003*. ACM SIGGRAPH, June 2003.
- [4] Alex Mendez Feliu and Mateu Sbert. Reusing frames in camera animation. In *Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2006)*, Plzen, Czech Republic, January 2006.
- [5] Sergei Zhukov, Andrei Iones, and Grigoriy Kronin. An ambient light illumination model. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop '98)*, pages 45–56. Springer Wien, 1998.
- [6] Andrei Iones, Anton Krupkin, Mateu Sbert, and Sergey Zhukov. Fast, realistic lighting for video games. *IEEE Computer Graphics and Applications*, 23(3):54–64, May/June 2003.
- [7] Per H. Christensen. Course 27: Global illumination for interactive applications and high-quality animations. In Cyrille Domez and Philipp Slusallek, editors, *ACM SIGGRAPH 2003 Full Conference DVD-ROM*. ACM SIGGRAPH, 2003.
- [8] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. In *Computer Graphics (ACM SIGGRAPH '84 Proceedings)*, volume 18, pages 137–145, July 1984.
- [9] Alex Mendez Feliu and Mateu Sbert. Combining light animation with obscurances for glossy environments. *Computer Animation and Virtual worlds*, 15(3-4):463–470, July 2004.
- [10] Alex Mendez, Mateu Sbert, and Jordi Cata. Real-time obscurances with color bleeding. In *Proceedings of Spring Confer-*

---

```

for all l in lights do
    ambient[l]=computeAmbientIntensity(l)
for i = 1 to numFrames do
    currentEye = getEye(i)
    for all pixel in images[i] do
        hit=traceRay(currentEye, pixel)
        obs=getObscurances(hit)
        indirectcolor=obs*hit.diffuse
        if isSpecular(hit) then
            specularcolor=getSpecularColor(hit)
        for all l in lights do
            directcolor[l]=getDirectLight(hit,l)
        for k = (i- $N_r$ ) to (i+ $N_r$ ) do
            if i==k then
                setImageValues(images[k], directcolor, indirectcolor, specularcolor)
            else
                reuseValuesInImage(images[k], directcolor, indirectcolor)
        computeFinalImage(i- $N_r$ )
    for i=(numFrames- $N_r$ +1) to numFrames do
        computeFinalImage(i)

```

---

Algorithm 1: The algorithm for the combination of camera and light animation.

*ence on Computer Graphics (SCCG 2003)*, Bratislava, Slovakia, April 2003.

- [11] Philippe Bekaert, Mateu Sbert, and John Halton. Accelerating path tracing by reusing paths. In *Rendering Techniques 2002 (Proceedings of the Thirteenth Eurographics Workshop on Rendering)*, June 2002.
- [12] Mateu Sbert, Francesc Castro, and John Halton. Reuse of paths in light source animation. In *Proceedings of Computer Graphics International 2004 (CGI '04)*, pages 532–535. IEEE Computer Society, June 2004.
- [13] Mateu Sbert, Laszlo Szecsi, and Laszlo Szirmay-Kalos. Real-time light animation. *Computer Graphics Forum (Eurographics 2004 Proceedings)*, 23(3), September 2004.

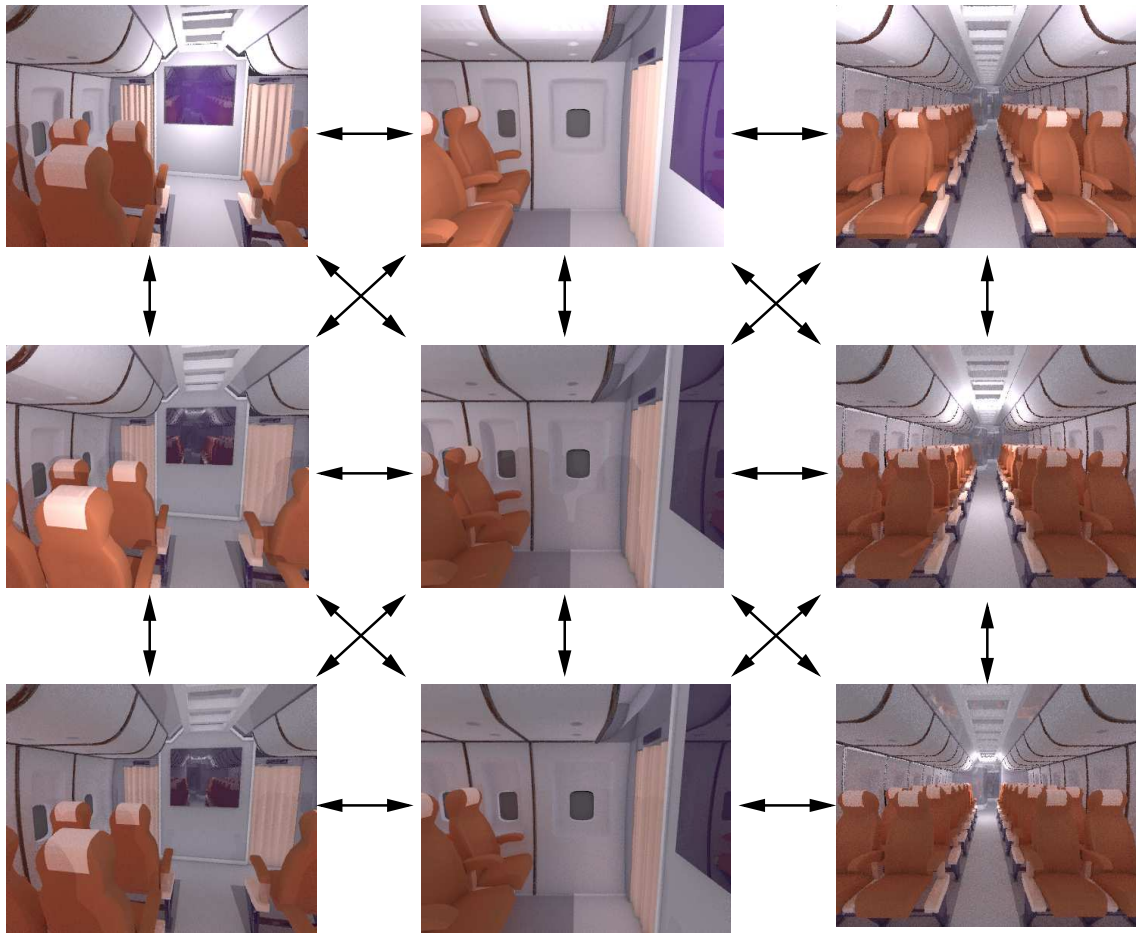


Figure 5: Here we see a few frames of the animation of the aircraft model. The arrows show a few of the multiple possibilities we have to generate different movies with our set of images.