Fast, Realistic Lighting for Video Games

Andrey Iones and Anton Krupkin
Saber Interactive

Mateu Sbert
University of Girona, Spain

Sergey Zhukov
Creat Studio

A novel, view-independent technology produces natural-looking lighting effects faster than radiosity and ray tracing. The approach is suited for 3D real-time interactive applications and production rendering.

Usually you would use an expensive method like radiosity or ray tracing (or a combination of both) to simulate global illumination effects produced by diffuse interreflections. These powerful methods can produce photorealistic images but are time-consuming, require large amounts of memory, and are difficult for a nonexpert to control. For these reasons, graphic artists performing production rendering rarely use radiosity-like global illumination methods.[1] In practice, moreover, artists frequently need to render images that aren't photorealistic but rather "bend and shape the reality to the will of the [movie] director."[1] Such an approach works well for filmmaking, commercials, game production, and VR systems. Such applications need a method to quickly reproduce lighting effects that has the power of, but avoids the expense of, radiosity computations.

We present a novel technology that simulates natural-looking lighting effects. Three-dimensional real-time applications and production quality rendering can both use this technology in a similar way. Our method simulates global illumination effects using the ambient light illumination model.[2] This model estimates ambient light more accurately than Phong and other local illumination models but without the expense of radiosity. The model uses the so-called obscurance coefficients of scene points to estimate ambient light. Obscurance, first introduced by Zhukov, is a local and purely geometric property of a scene point that resembles the integrated weighted form factors computed for some neighborhood of a given point.[2] Our empiric approach rapidly computes the illumination solution and renders believable and realistic looking images.

Our method stores the illumination computation results in lightmaps or obscurance map textures. These textures help modulate the scene's base textures. Storing illumination data in texture maps enables visualization of fine shading details without explicit meshing, resulting in faster rendering.

We were motivated by practical needs in developing the technology. End-users will find our proposed solutions easy to implement and convenient to control. For example, all parameters are intuitive and directly result in visual changes of produced images. Also, the algorithms don't require user intervention or impose modeling constraints and will work robustly on complex production models for illumination computations. All the techniques and algorithms described here complement our production renderers (Avid's Softimage and mental ray). Our lighting technology is now used in many challenging Creat Studio projects, such as 3D video games and computer animations (intros and cut scenes) for *Heretic-II*, *Civilization-II: Call To Power*, *Wu Tang*, and *Star Trek*.

Our technology simplifies and speeds up the production pipeline of both real-time applications and prerendered graphics. Because we based our illumination technique on ambient light estimation, the method accentuates scene geometry even without light sources. This enables rapid computing of images and requires fewer light sources to obtain the visuals. This decreases both light source setting and rendering time of image sequences.

Similarly to radiosity, our lighting solution is view-independent. Once computed, obscurance texture maps make it possible to rapidly reevaluate the illumination solution for moving light sources and for scenes with animated objects. Finally, because of the computation's data locality, the method allows a straightforward parallelization.

Our lighting technology is based on two principal issues addressed in the "Previous Work" sidebar.

Subdividing scenes into patches

Like radiosity, our model generates discrete patches for the scene. This problem is well addressed in the literature.[3-4] Most of these works discuss the scene subdivision problem for storing illumination results in

Previous Work

Two principal issues addressed in the lighting technology literature have influenced our work in terms of illumination computation and storage of computed data for efficient rendering.

Illumination computation methods

The principal goal of realistic image synthesis is to develop methods that allow realistic visualization of 3D scenes. To achieve this goal, researchers have developed numerous illumination models. The first models developed 30 years ago are local illumination models that are easy to implement and fast to compute. Later, much more complex global methods were devised that enable the generation of photorealistic-quality imagery.[1] The latter approaches range from classic radiosity and ray tracing schemes to more modern approaches.[2-5] The other way to create visually plausible images (that aren't photorealistic) is based on more ad hoc techniques such as negative intensity lights, non-ray-traced lights, and procedural shaders (in mental ray[6] or RenderMan[7]). These approaches strive to recreate the most appealing and believable lighting effects and might sacrifice physical realism for this goal. Such methods prove especially useful in 3D animation packages for rendering computer-generated imagery, where it's frequently desirable to use photosurrealistic techniques,[8] and in 3D real-time applications.

The literature has discussed several relevant methods for computing images by analyzing local scene geometry. Miller exploited an empirical approach called accessibility shading, where similar geometric properties were used to recreate the visual effects of aging of materials.[9] Martin et al. used transmittance textures to speed up hierarchical radiosity computation.[10] This work used transmittance textures to store information on how a polygon (receiver) is occluded with respect to another polygon (shooter) and modulates illumination textures that were computed without visibility information. Castro et al. compute an extended ambient term by classifying the surfaces into six classes according to their normals and solving a simplified radiosity system.[11] Zhukov first introduced obscurances, which are the core of our lighting technology, and their use resembles the accessibility shading one.[12] They are stored in obscurance maps (lightmaps) and simulate indirect diffuse illumination on a point (patch) through a kind of form-factor computation in the vicinity of the point (patch).

Illumination solution storage

To perform illumination computation, we subdivide the scene into patches. Hence, it seems natural to explicitly subdivide the original polygons into smaller ones and assign the computed illumination to their vertices.[1] To render the image, we apply a conventional color interpolation scheme to the polygons. A disadvantage of explicit subdivision is that it significantly increases the overall polygon count, incurring higher rendering time and storage requirements. This slowdown might not matter if you need to compute a radiosity solution for a static scene and render a few frames. In practice, it's often necessary to render animation sequences of hundreds or thousands of frames, so rendering time of already lit scenes becomes crucial, hence increasing the polygon count becomes undesirable.[13] The same situation happens in 3D real-time applications because frame rate primarily depends on scene complexity.

One approach to solving this problem is based on using lightmap textures.[14-16] These textures are lower resolution than the scene's base textures, and each lightmap texel corresponds to a patch created at scene subdivision for illumination computation. Therefore, lightmap textures allow storing multiple light samples for large polygons. This is in contrast to explicit scene subdivision with interpolative polygon shading, where color samples are stored in vertices

continued on p. 56

polygon (patch) vertices. Because we use texture maps to store illumination data, the subdivision occurs in a slightly different manner. We assume that the scene is represented by a collection of triangles.

The main part of subdividing scenes into patches is polygon clustering. Each cluster contains a number of polygons, all of which can be mapped into a 2D mapping space so that the mapping becomes a triangulation. We start with the creation of a polygon connectivity data structure for all polygons in the scene and then perform the breadth-first search algorithm to form clusters. We add a polygon P to the cluster if

- it's adjacent to some other polygon P' within the cluster;
- the angle between normal(P) and normal(P') is less than a certain predefined discontinuity angle (the notion of discontinuity angles is widely used in commercial ray tracers; often it's equal to 60 degrees;[5-6]
- it doesn't intersect any other cluster polygon when it's mapped into the mapping space; and
- it satisfies the clustering criterion, which we will describe.

You can choose clustering criteria based on different clustering ideas. Two options prove effective:

- Polygons with similar normals are clustered. The mapping space becomes a plane, and the mapping function is the projection onto this plane. When we add the first polygon to the cluster, we select the world-coordinate-system axes with the direction closest to polygon normal. We add polygon P if it passes a discontinuity test with this axis.
- Polygons covered by a continuous texture mapping are clustered.

The second choice usually yields excellent results provided that the scene is evenly textured. The first choice is best for nontextured scenes. These methods can be used together and selected based on other heuristics.

Once we create the polygon clusters, we subdivide them into patches. Each cluster has a mapping into a 2D mapping space, which the clustering algorithm defines. A regular grid tessellates this 2D space; each grid cell becomes a patch and hence a texel on a lightmap tex-

continued from p. 55

only. For lightmap computation the patches are still generated, but this occurs implicitly (virtually). Once illumination is computed and stored in textures, patches are dropped, and the polygon count doesn't increase at rendering time. In a related way, some work has used textures to represent the precomputed radiosity[13] or sampled illumination values.[17,18] Bastos et al. provides an application for these methods to speed up walkthrough of static scenes.[19]

Lightmap textures help modulate the scene's base textures, recreating fine lighting details. Graphics acceleration hardware supports this process, making lightmaps quite useful in 3D real-time applications. *Quake*, developed by id Software, provides an example of lightmapping.

In separate works, Arvo and Ward et al. introduced a related technique called illumination maps in the context of ray tracing to avoid recomputing the diffuse illumination when the viewpoint changes.[17,20] Commercially available renderers such as Autodesk's Lightscape and Integra's Inspirer employ illumination maps. These systems explicitly subdivide the scene into smaller polygons and store illumination maps in polygon vertices. Another approach, first introduced in Jensen and Christensen, is to store illumination information in photon maps using kd-trees.[5]

A possible drawback to using lightmaps instead of explicit scene subdivision into polygonal patches is that lightmap texture density is constant over the polygon. In practice, this drawback is unimportant. After all, you can subdivide large polygons into a few smaller ones and use more lightmap texels in polygons with a high variance of illumination, thus trading texture memory for image quality.

References

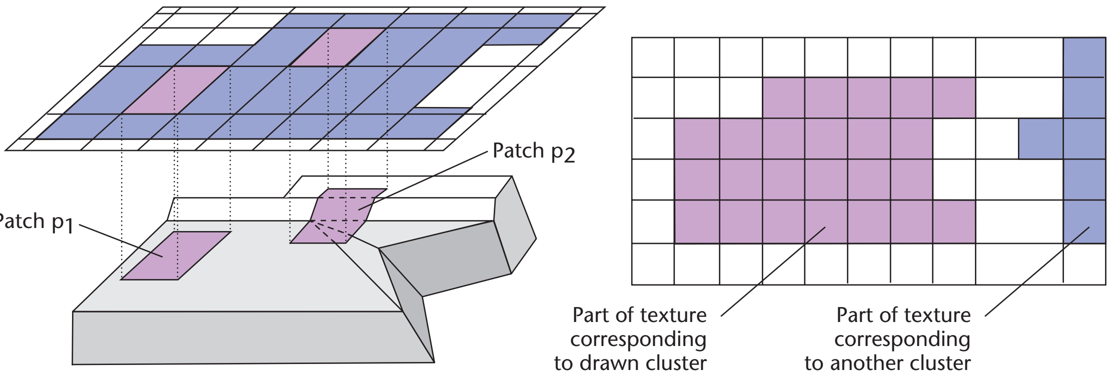
1. A.S. Glassner, *Principles of Digital Image Synthesis*, Morgan Kaufmann, 1995.
2. M. Cohen and J. Wallace, *Radiosity and Realistic Image Synthesis*, Academic Press Professional, 1993.
3. F. Sillion and C. Puech, *Radiosity and Global Illumination*, Morgan Kaufmann, 1994.
4. P. Bekaert, *Hierarchical and Stochastic Algorithms for Radiosity*, doctoral dissertation, Dept. of Computer Science, Katholieke Universiteit Leuven, Netherlands, 1999.
5. H. Jensen and N. Christensen, "Photon Maps in Bidirectional Monte Carlo Ray Tracing of Complex Objects," *Computers and Graphics*, vol. 19, no. 2, 1995, pp. 215-224.
6. *Softimage 3D Version 3.7 mental ray Programmer's Guide*, 1996.
7. S. Upstill, *The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics*, Addison-Wesley, 1992.
8. A. Apodaca, "Photosurrealism," *Proc. Eurographics Rendering Workshop*, Springer-Wien, 1998, pp. 315-322.
9. G. Miller, "Efficient Algorithms for Local and Global Accessibility Shading," *Proc. Siggraph*, 1994, pp. 319-326.
10. I. Martin, X. Pueyo, and D. Tost, "A Hardware Based Method for Hierarchical Radiosity," *Proc. Eurographics*, vol. 17, no. 3, 1998, pp. 159-164.
11. F. Castro, L. Neumann, and M. Sbert, "Extended Ambient Term," *J. Graphics Tools*, vol. 5, no. 4, 2000, pp. 1-7.
12. S. Zhukov, A. Iones, and G. Kronin, "An Ambient Light Illumination Model," *Rendering Techniques—Proc. Eurographics Rendering Workshop*, Springer-Wien, 1998, pp. 45-55.
13. K. Myszkowski, and T. Kunii, "Texture Mapping as an Alternative for Meshing during Walkthrough Animation," *Rendering Techniques—Proc. Eurographics Rendering Workshop*, Springer-Wien, 1994, pp. 389-400.
14. S. Collins, "Adaptive Splatting for Specular to Diffuse Light Transport," *Proc. 5th Eurographics Workshop on Rendering*, Springer-Wien, 1994, pp. 119-135.å
15. *Softimage 3D Version 3.7 Release Notes*, 1997.
16. S. Zhukov, A. Iones, and G. Kronin, "On a Practical Use of Light Maps in Real-Time Applications," *Proc. Spring Conf. Computer Graphics*, Comenius Univ., 1997
17. J. Arvo, "Transfer Equations in Global Illumination," *ACM Siggraph Course Notes*, vol. 42, ACM Press, 1993.
18. P. Heckbert, "Adaptive Radiosity Textures for Bidirectional Ray Tracing," *Computer Graphics, Proc. Siggraph*, vol. 24, 1990, pp. 145-154.
19. R. Bastos, M. Goslin, and H. Zhang, "Efficient Radiosity Rendering Using Textures and Bicubic Reconstruction," *Proc. ACM Symp. Interactive 3D Graphics*, ACM Press, 1997, pp. 71-74.
20. G Ward, F. Rubinstein, and R. Clear, "A Ray Tracing Solution for Diffuse Interreflection," *Proc. ACM Siggraph*, vol. 22, no. 4, 1988, pp. 85-92.

ture (see Figure 1). We select the grid's density according to the desired density of lightmap textures. We define a patch as a collection of pieces of polygons in the cluster mapped onto a single grid cell. Figure 2 illustrates this process for a sphere.

For obscurance and other illumination computations we need to obtain a patch center and the normal at that point. Often a patch has only one face mapped onto the corresponding texel. In this case, the patch center is well defined, and the patch normal becomes a smooth surface normal at that point. If several faces correspond to a patch, we compute the patch center as an appropriate average point.

Because our technique provides shared patches along polygon boundaries and filtering of adjacent texel intensities after computation, no abrupt shading changes occur along polygon edges and the images look smooth. Another advantage of our method is a patch might actually cover a large number of polygons in regions where the geometric resolution is higher than the required illumination resolution. Since only a few patches are generated in such regions, the illumination algorithm's complexity is much lower compared to the case of generating a patch for each polygon. We can easily increase patch resolution in places where it's not high enough.

Once we subdivide all scene clusters into patches, we pack the generated lightmap parts into larger rectangular textures. Therefore, a single lightmap texture corresponds to a large number of polygon clusters, as Figure 1a shows. This is unlike other implementations that cre-


Patch p2
Patch p1
Part of texture corresponding to drawn cluster
Part of texture corresponding to another cluster

1 **Subdividing a scene into patches. (a) Part of the lightmap texture corresponding to a cluster of polygons appears on the mapping plane and (b) final lightmap texture.**

ate lightmap textures individually for each polygon or for polygon stripes.[7-8] Although such an approach allows creation of lightmaps more adaptively, it inefficiently uses texture memory because only part of a rectangular texture is covered by useful texels. Also, in practice it's undesirable to have a huge number of small textures from a hardware texture management point of view.

Models

In this section we present our ambient light illumination model based on obscurances.

Obscurance illumination

Most local illumination models account for diffuse inter-reflections using ambient light that's crudely approximated as constant over the whole scene. In reality, in diffusely illuminated scenes composed of mostly diffuse surfaces, the illumination is not constant. For example, in a diffusely lit room the illumination changes over wall surfaces—it's darker near room corners. A similar effect is a shadow under a car on a cloudy day. We can visually approximate such lighting (or darkening) effects in obscured areas. Our obscurance-based illumination model emphasizes corners. Unlike radiosity, our model is nonphotorealistic but obtains visually pleasant images.

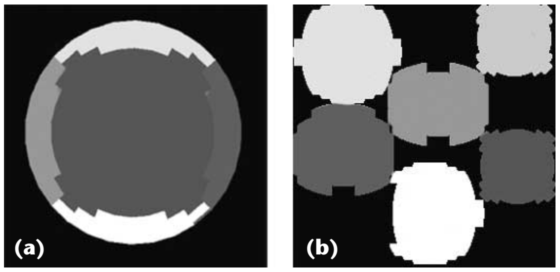
Roughly speaking, obscurance measures the part of the hemisphere obscured by neighbor surfaces. For example, near a corner of a room the patch obscurance is higher than in the center of the room. From a physics of light transport viewpoint, obscurance expresses the lack of secondary (reflected) light rays coming to the specific parts of the scene, thus making them darker. This differs from radiosity, which accounts for secondary reflections to increase intensity.

Empirical obscurance indirect illumination

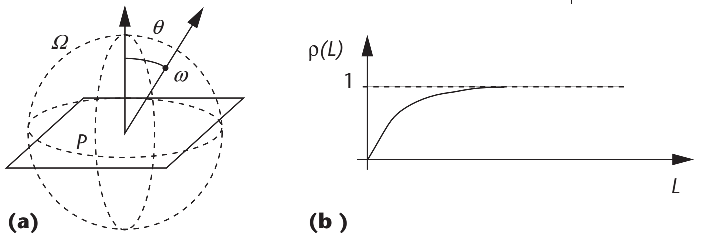
Let's assume that a scene has no specific light sources—in other words, perfectly diffuse light coming from everywhere illuminates the scene. The ambient light intensity for the whole environment is a direction-independent constant I_A (this is the same constant as used in other local illumination models). We introduce obscurances to avoid the disadvantages of the classic ambient term. Our basic approach is a distance-dependent ambient effect using a function ρ with values between 0 and 1 to represent the incoming ambient light.

Let P be a point on the surface in the scene and ω a direction in the normal hemisphere Ω with center P, aligned with the surface normal at P and lying in the outer part of the surface, see Figure 3a. We look for a function $\rho(L(P, \omega))$ to represent the obscurance for a point P in direction ω—that is, the magnitude of ambient light incoming from direction ω. L is the distance to the first intersection in this direction. (The name obscurance is a bit confusing here because it actually denotes the openness of a point, but we use the earlier introduced name.[2] This function should have the following (intuitive) properties:

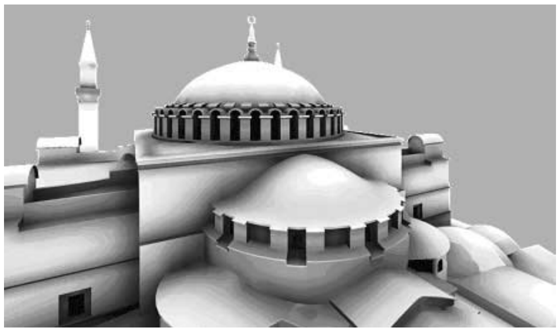
$$\rho(L) = \begin{cases} 0, \text{ for } L = 0 \\ 1, \text{ for } L = +\infty \end{cases}$$



2 **Subdividing a sphere into patches using polygons with similar normals. (a) Sphere mapped into a part of the lightmap texture corresponding to a cluster of polygons on the mapping plane and (b) final lightmap texture.**



3 **(a) Illustration of the notation in Equation 3. (b) Plot of the function ρ(*L*).**


4 **Obscurance map for a temple scene.**

$$\rho'(L) = \begin{cases} > 0 \\ 0, \text{for } L = +\infty \end{cases}$$

$$\rho''(L) < 0 \qquad (1)$$

That is, we look for a monotonous, increasingly smooth function (the farther the intersection, the higher the ambient term's contribution in this direction), a null value at intersection distance zero (fully occluded) and asymptotical value 1, and a monotonous decreasing derivative—see Figure 3a.

An evident family of functions that fits Equation 1 is

$$\rho(L) = 1 - e^{-\tau L} \qquad (2)$$

where τ is a positive parameter.

We now define the obscurance of point P as follows:

$$W(P) = \frac{1}{\pi} \int_{\omega \in \Omega} \rho(L(P,\omega)) \cos\theta d\omega \qquad (3)$$

Thus, the obscurance of a point P is the weighted average length of a chord originating from the point (we measure the chord length between P and the first intersection point with a surface in the scene). Clearly, for any surface point P: $0 \le W(P) \le 1$. For example, if P stands at the base midpoint of a hemisphere with radius R, then $W(P) = \rho(R)$. The function $W(P)$ reflects the local geometric properties of point P. Obscurance value 1 means that the point is fully open, while a value of 0 means the point is fully closed (this might happen only in the degenerate cases). Now, we assume that the more open a point, the greater its intensity (brightness). This reflects the fact that normally ambient lighting of a given point is primarily affected by its neighborhood. This is especially true for scenes without bright light sources that could affect the illumination at large distances. We write then for the reflected intensity at point P:

$$I(P) = I_A k_A(P) W(P) \qquad (4)$$

where $k_A(P)$ is the diffuse reflectance for ambient light. Interestingly, we can interpret the obscurance model with the $\rho = (1 - e^{-\tau L})$ function as the illumination at the nonreflecting boundaries of a nonscattering gas with opacity τ and constant volume emittance I_A.[9] If point P is totally unobscured ($\rho() \equiv 1$ over the whole hemisphere), then $I(P)$ becomes $k_A * I_A$.

Equation 4 actually determines the illumination model working in the absence of light sources. The obscurance of a patch is the average obscurance value over all its points. Figure 4 illustrates typical obscurance values for different patches. We will show that patch obscurance eventually resembles a patch's average form factor.

Figure 4 shows the obscurance map for a temple scene. Assuming all cases of reflectance are equal, this scene's illumination by ambient light (without specific light sources) using Equation 4 would give a similar image (up to a constant factor). A number of pseudoshadow effects are clearly visible. We claim that the scene looks similar to the view on a cloudy day. Using obscurances outlines the surface profile without any light source setting.

Obscurance is intrinsically geometric. It doesn't depend on light sources, so we only need to compute obscurance maps for a given scene and can store them for future use to recompute the lighting for moving light sources.

Ambient light illumination

The goal of our illumination model is to easily and quickly account for indirect illumination caused by diffuse interreflections. To compute the illumination of a given patch caused by a given light source, we separate the illumination into direct and indirect factors. We compute direct illumination using the standard local diffuse illumination model. Similar to ambient light, we empirically account for the indirect illumination from the light sources by using the obscurance coefficient. Specifically, we use the following equation (an extension of Equation 4):

$$I(P) = (I_A + I'_S(P)) * k_A(P) * W(P) + k_D(P) * I_S(P) \qquad (5)$$

where k_D is the diffuse reflection coefficient. I_S is the intensity of direct illumination from all visible light sources,

$$I_S(P) = \sum_{\substack{\text{by all visible} \\ \text{light sources}}} \frac{I_j}{r_j^2} \cos\alpha_j$$

I'_S is the intensity of indirect illumination coming from all light sources,

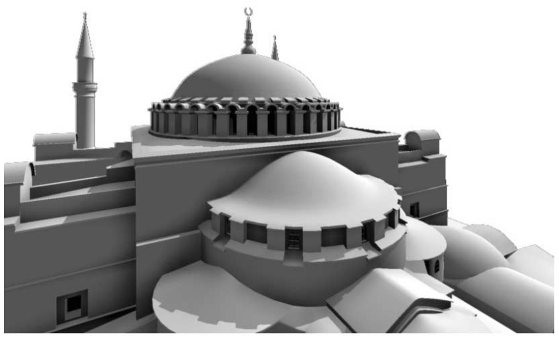
$$I'_S(P) = \beta * \sum_{\substack{\text{by all light} \\ \text{sources}}} \frac{I_j}{r_j^2}$$

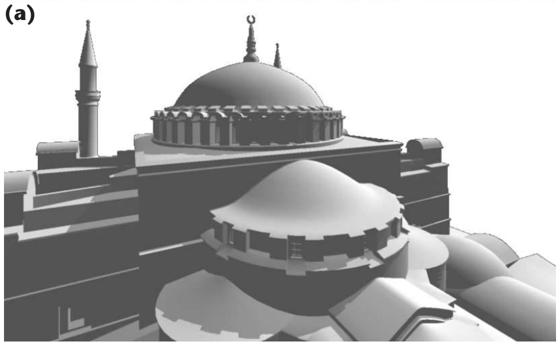
where I_j is the intensity of jth light source; r_j is the distance from light source j to patch P; α_j is the angle between direction toward the jth light source and patch normal; and β is a constant, $0 < \beta < 1$.

We account for indirect illumination from light sources by the term I'_S. The constant β scales down the indirect component with respect to the direct one. The indirect term I'_S doesn't contain cos() factors since we assume that secondary rays come to a patch from every direction. Note that all light sources are accounted for regardless of whether they are visible. Obviously, by not rejecting invisible light sources, we might get artifacts like lighting through an opaque wall. On the other hand, we properly account for a light source in a patch that is locally obscured by other surfaces.

Figure 5a shows a fragment of the same temple scene rendered using Equation 5 (with one light source). Although projected shadows appear, the overall impression differs only slightly from the scene rendered with only obscurances. Figure 5b shows the same scene rendered using conventional first-order ray tracing with constant ambient light. We can't properly discern the geometry of this scene and, therefore, the image is difficult to interpret—we need additional light sources to outline the profiles in the picture. For such purposes, we could use other ad hoc techniques such as using local lights with negative intensity to dim the scene, but these approaches are more expensive compared to using obscurances.

Our proposed technique uses significantly fewer light sources to obtain the desired lighting effects, decreasing both light source setting and ray tracing time.


(a)


(b)

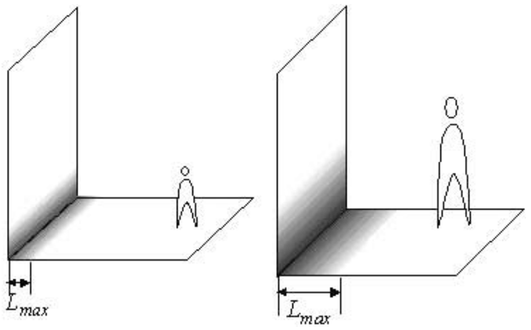
5 (a) Temple scene with one directional light source rendered using Equation 5. Apart from pseudoshadows, only projected shadows appear. (b) Scene rendered with first-order ray tracing with constant ambient light.

Limiting the maximum distance

From a practical point of view we consider only an $L_{\max}$ environment for the patch, where $L_{\max}$ is a scene-dependent parameter. This trades off the accuracy of taking into account all occlusions for the increase in speed, as we only have to consider a fraction of the scene for the computation of Equation 2. Function $\rho(L)$ becomes then $\rho(L) = 0$ for $L = 0$ and 1 for $L \geq L_{\max}$.

Experiments with different types of scenes show that $\tau = 1$ is a good choice in Equation 2. Thus, $L_{\max}$ is the primary parameter that controls the illumination model. However, there are some calibration issues related to $L_{\max}$. Assume that we need to compute illumination by ambient light for a scene composed of two perpendicular half planes (see Figure 6). We can't accomplish this task because the scene doesn't have any meters. So for practical purposes the size of the shadowed regions near the corner depends on the selection of the constant $L_{\max}$. This choice is based on the size of objects relevant to the scene (for example, avatars that will inhabit the scene in a VR application). Therefore, $L_{\max}$ controls the relative size of the shadowing effect. Assuming that a patch is several times smaller than a relevant object, we usually choose $L_{\max}$ to cover 10 to 100 patches.

In practice, we can select $L_{\max}$ rather loosely within certain reasonable limits. Once we compute an obscurance texture map, we edit its contrast and gamma in an image processing tool that changes shadow region shapes and sizes that otherwise $L_{\max}$ would control.



6 $L_{\max}$ selection depends on the width of the desired shadow region. Even though these and images differ, both are equally acceptable depending on the size of other relevant objects.

Computing obscurances

Obscurance coefficients are similar to integrated, weighted form factors. Indeed, for a point Q visible from point P on a given patch p, we define the differential form factor dF_{PQ} as

$$dF_{PQ} = \frac{1}{\pi r^2} \cos\theta_P \cos\theta_Q dS_Q dS_P$$

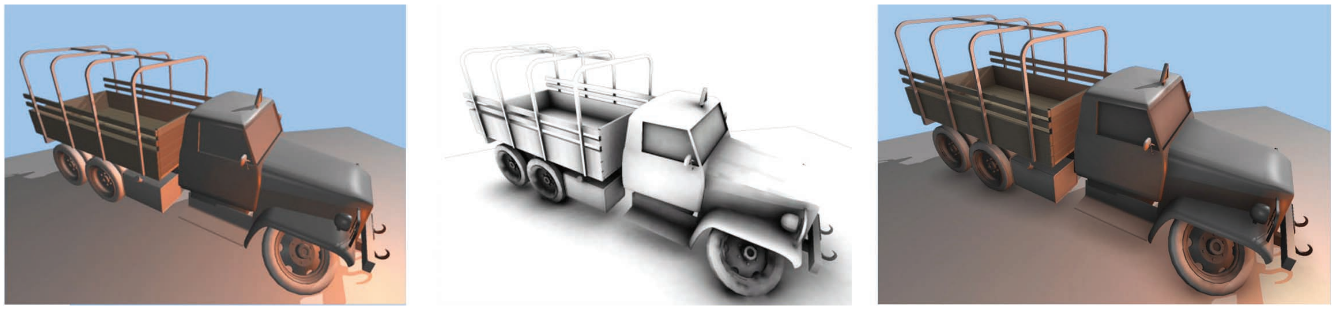
where θ_P, θ_Q are the angles between the line PQ and respective patch normals, dS_Q is the differential area at Q, and r is distance between P and Q.

Because $d\omega = 1/r^2 \cos\theta_Q dS_Q$ (using the definition of obscurance from Equation 3), we obtain for the obscurance of point P:

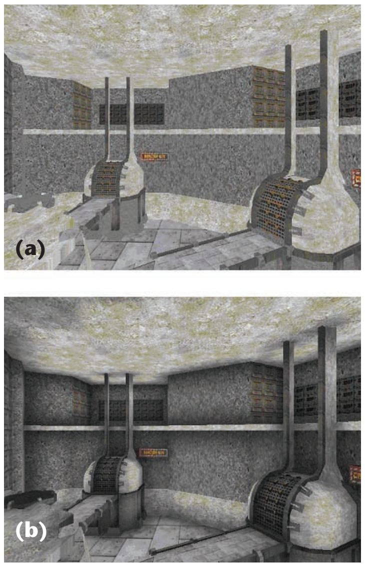
$$W(P) = \int_{\substack{\text{over all visible surface points} \\ \text{(differential patches } Q)}} \rho(\text{dist}(P,Q)) dF_{PQ} \qquad (6)$$

and taking the average over the area of patch p, we obtain the obscurance of p.

Equation 6 indicates the computation method for the



7 Automobile model (43,000 polygons) from *Civilization-III* video game cut scene: (a) ray traced with one light source and (b) obscurances map. (c) Final composed image with obscurances. Obscurance computation took 9 minutes.



8 Human environment (3,000 polygons) with (a) textures only and with (b) textures and obscurance maps. The geometry in (a) is more easily discernable than in (b). Obscurance computation took 45 seconds.

9 Alien environment with textures and obscurance maps.

obscurance of a given patch. To compute the obscurance, you could use, for example, the hemicube method. In this case, we place a single hemicube over the patch center—that is, we approximate the obscurance of a patch with the obscurance of the patch center. Since the function $\rho(L)$ is less than 1 only in the range $[0 .. L_{\max}]$, we might use only patches in the $L_{\max}$ neighborhood of P. We therefore need a slight modification of the hemicube method to account for missing patches—initializing pixels to 0—and weight differential form factors using $\rho(L)$—storing weighted with $\rho(L)$ values instead of differential form factors. The locality of the obscurance method makes it much faster than conventional radiosity. We can also apply other form factor computation techniques, such as Monte Carlo (see, for instance, Bekaert[10]), to compute obscurances.

Ambient light illumination analysis

We consider here the cost and complexity of obscurance computation using the hemicube approach from our implementation. Other algorithms to compute obscurances might exhibit different cost and complexity. Suppose we fixed the hemicube resolution and the parameter $L_{\max}$. Let n_p be the number of patches and n_s the polygons in the scene. We distinguish two different complexities.

To compute obscurance for a patch, we must project all polygons in the $L_{\max}$ neighborhood onto the hemicube. If we consider this neighborhood as defined by a constant fraction k of scene volume ($L_{\max}$ is the same for all patches), then the same fraction of scene polygons is always projected. This produces a cost of $O(k * n_p * n_s)$, with k quite small. If we increase patch resolution without increasing the number of polygons, the cost will grow as $O(n_p)$. However, if we increase the number of polygons retaining the same patch resolution, the cost will grow as $O(n_p * n_p)$, but the constant is small. This small constant implies that our approach outperforms roughly linear radiosity methods, such as hierarchical radiosity with clustering,[11] in fairly large scenes. Despite its lower asymptotic complexity, hierarchical radiosity isn't feasible for scenes with millions of patches.

On the other hand, it's not fair to compare a purely geometrical quantity computation with the determination of the flux of energy density (radiosity). We should compare obscurance computation with its peer: form factor computation. Obscurances retain from form factors the capability of generating high-quality soft shadows, and we don't need to recompute them for a change in illumination settings. Clearly, obscurance computation outperforms form factor computation, with no need to consider the quadratical order in form factor storage, while the memory required for obscurance maps is linear relative to the number of patches. For scenes used in 3D interactive applications of roughly 20,000 polygons, the typical memory requirements

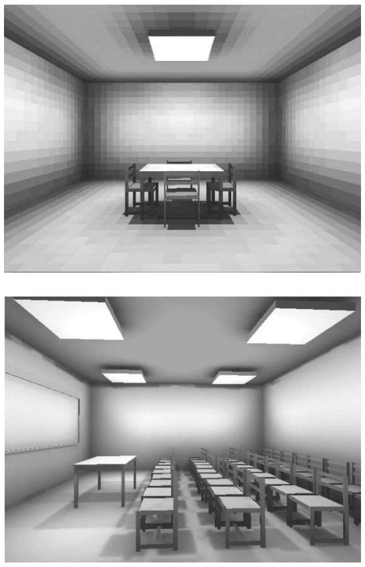

10 Shirley's scenes computed with the obscurance method. (Scene models courtesy of Peter Shirley.)

Table 1. Table and chair scene from Figure 10 (180 polygons).

Method	Number of Patches	Time (seconds)
Hierarchical radiosity	23,000	480
Hierarchical Monte Carlo radiosity	1,777	20
	17,617	320
Multipath Monte Carlo radiosity	12,000	1,100
Obscurance	2,000	1
	15,000	6

Table 2. Classroom scene from Figure 10 (1,244 polygons).

Method	Number of Patches	Time (seconds)
Hierarchical radiosity	18,000	450
	60,000	Ran out of memory (> 64 Mbytes)
Hierarchical Monte Carlo radiosity	77,202	570
Obscurance	60,000	240

for storing all lightmap textures are about five 256 × 256 texel textures.

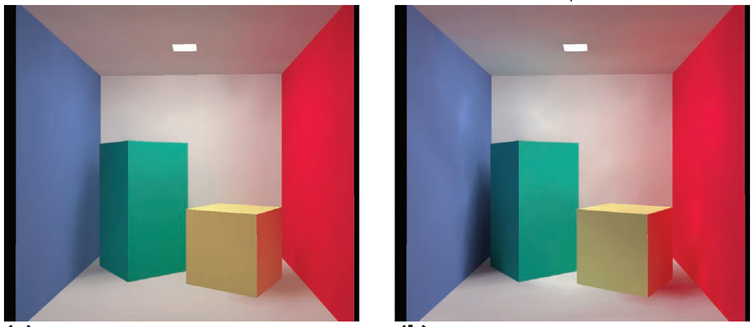
Comparison with other methods

Our initial motivation in developing our ambient light illumination model was the need to compute scene illumination in our real-time 3D games, where scene complexity ranges up to 20,000 polygons. Currently, our lighting tool can handle complex scenes composed of up to 400,000 polygons subdivided into a million implicit patches. We created the scenes shown in Figures 7, 8, and 9 at Creat Studios using Softimage for our animation projects on a Pentium-200 machine. We used a large patch count since we intended the scenes for close-up rendering.

To compare our method's performance with other approaches, we computed the illumination in Peter Shirley's scenes. Figure 10 shows the test scenes computed using our method.

Tables 1 and 2 present comparisons of our method with hierarchical radiosity, hierarchical Monte Carlo radiosity,[10] and multipath Monte Carlo radiosity.[12] The timings are given for a Pentium-200 or similar machines. The images produced using hierarchical Monte Carlo and multipath Monte Carlo radiosity methods with the timings given are noisy. To obtain high-quality images, you must significantly increase the timings because the methods converge slowly. Obscurance timings in both scenes correspond roughly to $O(k * n_p * n_s)$ complexity. Apparent differences in this behavior, such as between outdoor (Figure 7) and indoor (Figure 8) scenes are caused by a different L_{max} value selection, which in turn gives a different k value.

We don't compare our approach with the radiosity method in terms of accuracy. Indeed, the ultimate goal of our model is to obtain visually pleasant, realistic-looking lighting effects, not to solve radiosity system issues. The primary justification for our approach's utility is that the images produced look believable and realistic. Nevertheless, at its best, obscurances mimic radiosity's quality when light sources are positioned near the ceiling and light most



11 Cornell box scene computed with (a) the obscurance method and (b) with shooting random walk radiosity.

of the scene (see Figure 11). Conversely, when the light source points against and is near a wall—that is, when there is an important secondary reflector—obscurances poorly imitate radiosity, see Figure 12 (next page). Considering these important secondary reflectors as additional light sources such as in progressive radiosity[3] could solve this problem.

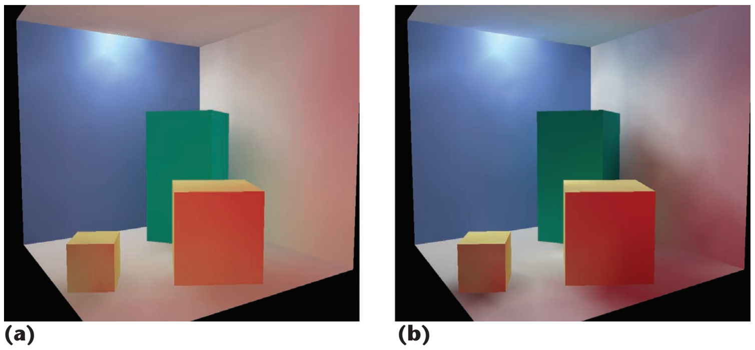
Implementation

Our stand-alone lighting tool computes obscurances or offers a full illumination solution for static scenes. Computation results are stored in texture maps assigned to the given model.

3D real-time applications

If a scene is intended for use in a real-time application, the system stores and computes the scene's overall illumination with light sources in lightmap textures. Then, at runtime, lightmap textures help modulate (add to or multiply) the scene's base textures.

Our described technology speeds up the production pipeline. The technology's application to our game envi-



12 **Cornell box scene with light pointing to a wall. Computed with (a) obscurance method and (b) hierarchical Monte Carlo radiosity. The images don't show the light sources.**

ronment development cycle consists of two stages: geometry modeling and lighting (shown in Figure 13). The crucial aspect in geometry modeling for games is that it requires an enormous (in the order of hundreds) number of iterations to achieve and fine-tune playability. Clearly, it's useless to spend much time texturing an unfinished model, and this is where our technology can assist. We compute scene illumination using ambient light only (without any light source setting) and apply the computed lightmaps for the scene. This makes scene geometry easily discernable. That is not the case at all for a preliminarily textured scene (see Figure 8). As a result of the computational effectiveness and usefulness of the technology, obscurance computation became a standard step of model conversion from Softimage into the game format in our production process.

Once we finish the geometry, we texture the scene model and set light sources. Because illumination computation with precomputed obscurance maps is fast, a modeler could perform as many iterations as needed to carefully set and adjust light sources to achieve the desired lighting effects. In practice, for scenes containing up to 20,000 polygons–which are used in our real-time applications—this recomputation occurs at interactive rates.

High-quality rendering applications

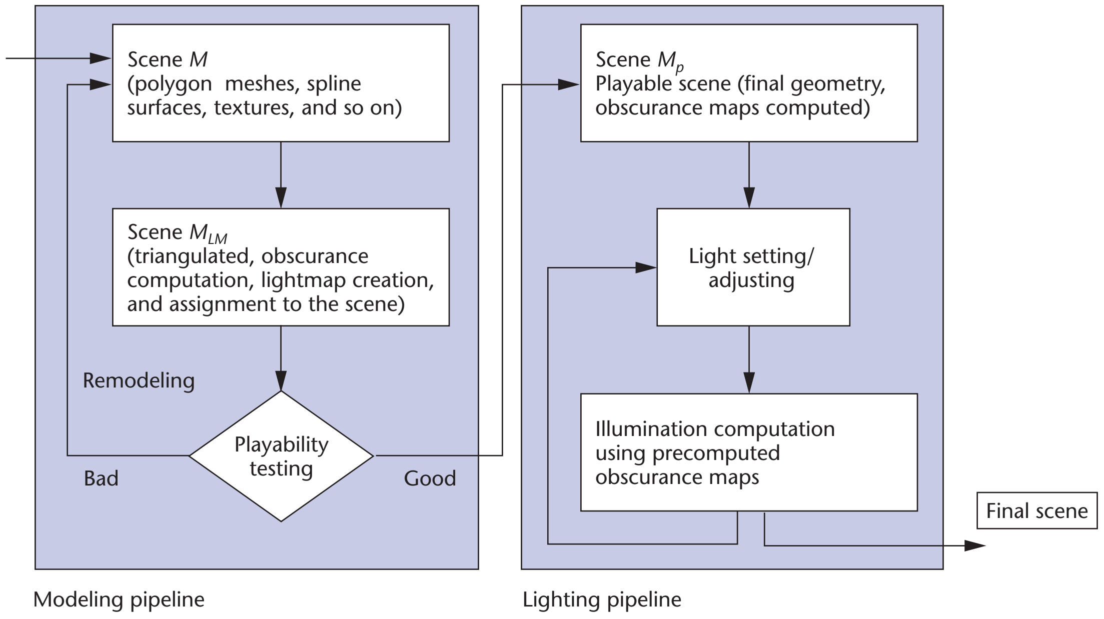
Let's discuss rendering an animation sequence of a camera walkthrough in a static scene. Our goal is to account for ambient light distribution using the ambient light illumination model in a standard ray-tracer rendering pipeline (in our case, Softimage and mental ray). For this purpose, we compute the obscurances that are stored in obscurance map textures assigned to the given Softimage model.

In our implementation, we wrote a material shader that describes the visible material of an object.[5] Such a shader executes on each hit of a ray with a surface. Instead of computing constant ambient light I_A, this shader computes its estimation, $I_A * W(P)$, where P is the hit point. Obscurance $W(P)$ is taken from the appropriate obscurance texture map. Implementing the ambient light illumination model via a material shader is the most accurate.

Using a standard ray tracer let us concentrate on developing the code we needed: scene subdivision into patches, obscurance computation, efficient storage of the data in texture maps, and assignment of textures to the given model. A standard ray tracer (mental ray) takes care of all important rendering tasks, including ray tracing itself, scene management, and antialiasing. Our technology can handle objects created with advanced modeling techniques such as spline surfaces and shape animation. Before rendering, we usually convert such models into polygon meshes. We then assign obscurance texture maps to these meshes, avoiding artifacts caused by incoherent meshing.

Conclusion

Currently, we use our technology in our game development projects, as well as on most of the animation projects at Creat Studio. Among them are intros, out-



Scene M
(polygon meshes, spline surfaces, textures, and so on)
Scene M_{LM}
(triangulated, obscurance computation, lightmap creation, and assignment to the scene)
Remodeling
Playability testing
Bad
Good
Scene M_p
Playable scene (final geometry, obscurance maps computed)
Light setting/ adjusting
Illumination computation using precomputed obscurance maps
Final scene
Modeling pipeline
Lighting pipeline

13 **Pipeline for game environment production.**

ros, and cut scenes for the Activision games *Heretic-II*, *Quake-II Ground Zero* mission pack, *Civilization-II:Call to Power*, and others. The use of our technology significantly improves image quality and reduces overall production time on various stages of project development. The "Key Features" sidebar summarizes the benefits our technology offers.

The technology presented in this article lays the groundwork for future development in different directions. To mention a few, we plan to improve the incorporation of the lighting tool in the production pipeline, thereby effectively performing obscurance recomputations in scenes with animation models. We must properly handle temporary aliasing artifacts in this scenario.

Because obscurances are similar to form factors, we can apply many ideas developed in radiosity research to obscurance computations. Specifically, improved adaptive, hierarchical clustering or Monte Carlo schemes can speed up computations, especially for large $L_{\max}$ values.

Overall, our realistic lighting simulation technology is suitable for applications that require sacrificing the physical accuracy of computations, but need speed and ease of use for light source setting, scene rendering, and overall production pipeline simplification. ■

Key Features

The key features and advantages of our technology include the following:

- Ambient light illumination model simulates nonconstant ambient light distribution in the environment without recourse to expensive global illumination methods.
- The ability to accentuate a scene profile without a single light source facilitates rapid computation of realistic images. This is important in the early stages of animation project development or when modeling environments for a game when the models aren't textured and model geometry isn't distinguishable without lighting.
- Use of fewer light sources to obtain the desired image decreases time for rendering at ray tracing and time spent by a modeler to set and adjust light sources.
- Our straightforward algorithms don't rely on complex data structures, making the technology easy to implement. This leads to lower multiplicative constants in big Oh bounds for time and space complexities, which are moderate for the obscurance computation algorithm compared to radiosity.
- The technology doesn't require user intervention or impose modeling constraints. It works robustly on complex production models that might not be well organized for illumination computations.
- Our view-independent lighting solution is useful for real-time applications, enabling the visualization of global illumination effects at interactive rates. For animation sequence rendering, using ray tracing and our solution improves visual quality.
- Once obscurance maps for the given static scene are computed, it's easy to recompute lighting for moving light sources.
- Similarly, if a scene has animated objects, recomputation of obscurance maps only occur in the neighborhood of the moving objects.
- The method's data locality enables a straightforward parallelization.
- The radiosity realistic effect of color bleeding can be easily incorporated in the obscurances method. Figures 11a and 12a in the main text illustrate the first results of our running research in this direction.

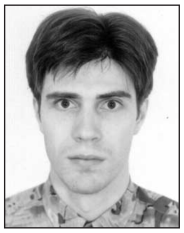
Acknowledgments

Mateu Sbert's research is partially supported by grants TIC-2001-2416-C03-01 from the Spanish government, 2001-SGR-00296 from the Catalan Government, and Spanish-Austrian Joint Action number HU2000-0011. We thank Laszlo Neumann for his useful comments. Anonymous IEEE *CG&A* reviewers helped to improve earlier drafts of this article.

References

1. A. Apodaca, "Photosurrealism," *Proc. Eurographics Rendering Workshop*, Springer-Wien, 1998, pp. 315-322.
2. S. Zhukov, A. Iones, and G. Kronin, "An Ambient Light Illumination Model," *Rendering Techniques—Proc. Eurographics Rendering Workshop*, Springer-Wien, 1998, pp. 45-55.
3. M. Cohen and J. Wallace, *Radiosity and Realistic Image Synthesis*, Academic Press Professional, 1993.
4. F. Sillion and C. Puech, *Radiosity and Global Illumination*, Morgan Kaufmann, 1994.
5. *Softimage 3D Version 3.7 mental ray Programmer's Guide*, 1996.
6. *Softimage 3D Version 3.7 Release Notes*, 1997.
7. K. Myszkowski, and T. Kunii, "Texture Mapping as an Alternative for Meshing during Walkthrough Animation," *Rendering Techniques—Proc. Eurographics Rendering Workshop*, Springer-Wien, 1994, pp. 389-400.
8. R. Bastos, M. Goslin, and H. Zhang, "Efficient Radiosity Rendering Using Textures and Bicubic Reconstruction," *Proc. ACM Symp. Interactive 3D Graphics*, ACM Press, 1997, pp. 71-74.
9. J. Arvo, "Backwards Ray Tracing," *ACM Siggraph Course Notes—Developments in Ray Tracing*, vol. 12, ACM Press, 1986.
10. P. Bekaert, *Hierarchical and Stochastic Algorithms for Radiosity*, doctoral dissertation, Dept. of Computer Science, Katholieke Universiteit Leuven, Netherlands, 1999.
11. F. Sillion, "A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters," *IEEE Trans. Visualization and Computer Graphics*, vol. 3, no. 1, 1997, pp. 23-38.
12. M. Sbert et al., "Global Multipath Monte Carlo Algorithms for Radiosity," *The Visual Computer*, vol. 12, no. 2, 1996, pp. 47-61.

Andrey Iones *is a CTO at Saber Interactive, New York. His research interests include video game design, rendering algorithms, integral geometry techniques applied to global illumination and collision detection, and artificial intelligence. Iones received an MS in applied mathematics and a PhD in computer science from the Technical University of St. Petersburg, Russia.*



Anton Krupkin *is a lead R&D person at Saber Interactive, New York. His research interests include video game design, rendering algorithms, hardware accelerated graphics, and integral geometry techniques applied to global illumination and collision detection. Krupkin received an MS in applied mathematics and computer graphics from the Technical University of St. Petersburg.*

Sergey Zhukov *is a production director at Creat Studio, St. Petersburg. His research interests include computer graphics, rendering algorithms, game design, artificial intelligence, and robotics. Zhukov received an MS in applied mathematics and a PhD in computer science from the Technical University of St. Petersburg.*

Mateu Sbert *is an associate professor in computer science at the University of Girona, Spain. His research interests include application of Monte Carlo techniques to radiosity and global illumination. Sbert received an MS in theoretical physics from the University of Valencia, an MS in mathematics from U.N.E.D. (National Distance University of Spain), Madrid, and a PhD in computer science from the Technical University of Catalonia, where he received the best PhD award.*

Readers may contact Mateu Sbert at Institute of Informatics and Applications, Campus Montilivi, Edifici PIV, Univ. of Girona, E-17071 Girona, Spain; mateu@ima.udg.es.

For further information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.