Indirect Diffuse and Glossy Illumination on the GPU

László Szirmay-Kalos and István Lazányi
Department of Control Engineering and Information Technology
Budapest University of Technology, Hungary
Email: szirmay@iit.bme.hu

Abstract

This paper presents a fast approximation method to obtain the indirect diffuse or glossy reflection on a dynamic object, caused by a diffuse or a moderately glossy environment. Instead of tracing rays to find the incoming illumination, we look up the indirect illumination from a cube map rendered from the reference point that is in the vicinity of the object. However, to cope with the difference between the incoming illumination of the reference point and of the shaded points, we apply a correction that uses geometric information also stored in cube map texels. This geometric information is the distance between the reference point and the surface visible from a cube map texel. The method computes indirect illumination although approximately, but providing very pleasing visual quality. The method suits very well to the GPU architecture, and can render these effects interactively. The primary application area of the proposed method is the introduction of diffuse and specular interreflections in games.

1 Introduction

Final gathering, i.e. the computation of the reflection of the indirect illumination toward the eye, is one of the most time consuming steps of realistic rendering. According to the rendering equation, the reflected radiance of point $\vec{x}$ in viewing direction $\vec{\omega}$ can be expressed by the following integral

$$L^r(\vec{x} \to \vec{\omega}) = \int_{\Omega'} L^{in}(\vec{x} \leftarrow \vec{y}(\vec{\omega}')) \cdot f_r(\vec{\omega}' \to \vec{x} \to \vec{\omega}) \cdot \cos\theta_{\vec{x}}\, d\omega',$$

where Ω' is the set of possible illumination directions, $L^{in}(\vec{x} \leftarrow \vec{y}(\vec{\omega}'))$ is the incoming radiance arriving at point $\vec{x}$ from point $\vec{y}$ visible in illumination direction $\vec{\omega}'$, f_r is the BRDF, and $\theta_{\vec{x}}$ is the angle between the surface normal at point $\vec{x}$ and the illumination direction.

The evaluation of this integral usually requires many sampling rays from each shaded point. Ray casting finds *illuminating points* $\vec{y}$ for *shaded point* $\vec{x}$ at different directions (Figure 1), and the radiance of these illumination points is inserted into a numerical quadrature approximating the rendering equation. In practice, number P of shaded points is over hundred thousands or millions, while number D of sample directions is about a hundred or a thousand to eliminate annoying sampling artifacts. On the other hand, in games and in real-time systems, rendering cannot take more than a few tens of milliseconds. This time does not allow tracing $P \cdot D$, i.e. a large number of rays.

To solve this complexity problem, we can exploit the fact that in games the dynamic objects are usually significantly smaller than their environment. Thus the global indirect illumination of the environment can be computed separately, since it is not really affected by the smaller dynamic objects. On the other hand, when the indirect illumination of dynamic objects is evaluated, their small size makes it possible to reuse illumination information obtained when shading its other points.

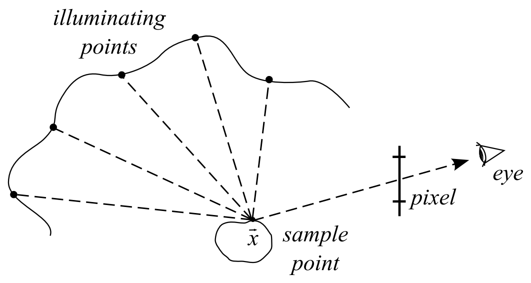


Figure 1: *Indirect illumination with sampling rays.*

The first idea of this paper is to trace rays with the graphics hardware just from a single reference point being in the vicinity of the dynamic object. Then the illuminating points selected by these rays and their radiance are used not only for the reference point, but for all visible points of the dynamic object. From a different point of view, tracing rays locates *virtual light sources* which illuminate the origin of these rays. We propose to find a collection of virtual light sources from a reference point, and then use these virtual light sources to illuminate all shaded points (Figure 2).

This approach has two advantages. On the one hand, instead of tracing $P \cdot D$ rays, we solve the rendering problem by tracing only D rays. On the other hand, these rays form a bundle meeting in the reference point and are regularly spaced. Such ray bundles can be very efficiently traced by the graphics hardware. On the other hand, this simplification also has disadvantages. Assuming that the same set of illuminating points are visible from each shaded point, self-shadowing effects are ignored. However, while shadows are crucial for direct lighting, shadows from indirect lighting are not so visually important. Thus the user or the gamer finds this simplification acceptable. Additionally, the used virtual light sources must be visible from the reference point. In concave environments, however, it may happen that an environment point is not visible from the reference point but may illuminate the shaded point. However, indirect illumination is usually quite smooth, thus ignoring smaller parts of the potentially illuminating surfaces does not lead to noticeable errors.

Unfortunately, this simplification alone cannot allow real time frame rates. The evaluation of the reflected radiance at a shaded point still requires the evaluation of the BRDF and the orientation angle, and the multiplication with the radiance of the illuminating point by D times. Although the number of rays traced to obtain indirect illumination is reduced from $P \cdot D$ to D, but the illumination formula must be evaluated $P \cdot D$ times. These computations would still need too much time.

In order to further increase the rendering speed, we propose to carry out as much computation globally for all shaded points, as possible. Clearly, this is again an approximation, since the weighting of the radiance of each illumination point at each shaded point is different. From mathematical point of view, we need to evaluate an integral of the product of the illumination and the local reflection for every shaded point. To allow global computation, the integral

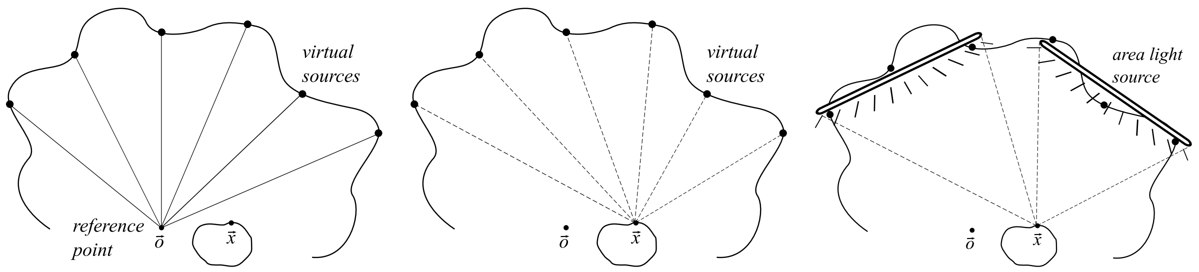


Figure 2: *The basic idea of the proposed method: first virtual lights sampled from reference point $\vec{o}$ are identified, then these point lights are grouped into large area lights. At shaded points $\vec{x}$ the illumination of a relatively small number of area lights is computed without visibility tests.*

of these products is approximated by the product of the integrals of the illumination and the local reflection, thus the illumination can be handled globally for all shaded points.

Intuitively, global computation means that the sets of virtual light sources are replaced by larger homogeneous area light sources. Since the total area of these lights is assumed to be visible, the reflected radiance can be analytically evaluated once for a whole set of virtual light sources.

2 Previous Work

Environment mapping[Blinn and Newell 1976] has been originally proposed to render ideal mirrors in local illumination frameworks, then extended to approximate general secondary rays without expensive ray-tracing[Greene 1984; Reinhard et al. 1994; Debevec 1998]. Classical environment mapping can also be applied for glossy and diffuse reflections as well. The usual trick is the convolution of the angular variation of the BRDF with the environment map during preprocessing [Ramamoorthi and Hanrahan 2001]. This step enables us to determine the illumination of an arbitrarily oriented surface patch with a single environment map lookup during rendering. A fundamental problem of this approach is that the generated environment map correctly represents the direction dependent illumination only at a single point, the reference point of the object. For other points, the environment map is only an approximation, where the error depends on the ratio of the distances between the point of interest and the reference point, and between the point of interest and the surfaces composing the environment (Figure 3).

One possible solution is to use multiple environment maps [G. et al. 1998; Zhou et al. 2005], which can be compressed using spherical harmonics [Ramamoorthi and Hanrahan 2001; Kristensen et al. 2005] or wavelets [Zhou et al. 2005]. For example, Greger et al.[G. et al. 1998] calculate and store the direction dependent illumination in the vertices of a bi-level grid subdividing the object scene. During run-time, irradiance values of an arbitrary point are calculated by tri-linearly interpolating the values obtained from the neighboring grid vertices. While Greger et al. used a precomputed radiosity solution to initialize the data structures, Mantiuk et al.[Mantiuk et al. 2002] calculated these values during run-time using an iterative algorithm that simulates the multiple bounces of light. Unfortunately, the generation and compression of many environment maps require considerable time which is not available during real-time rendering. Thus most of this computation should be done during preprocessing, which imposes restrictions on dynamic scenes.

Instead of working with many environment maps, another possible approach is to use a single or a few environment maps which are “localized”, so that a single map provides different illumination information for every point, based on the relative location from the reference point. GPU based localized image based lighting has been suggested by Bjorke[Bjorke 2004], where a proxy geometry (e.g. a sphere or a cube) of the environment is intersected by the reflection ray to obtain the visible point. The *Approximate Ray-Tracing* approach[Szirmay-Kalos et al. 2005], on the other hand, stores the distance values between the reference point and the environment and applies an iterative process to identify the real hit point of those rays that are not originated in the reference point.

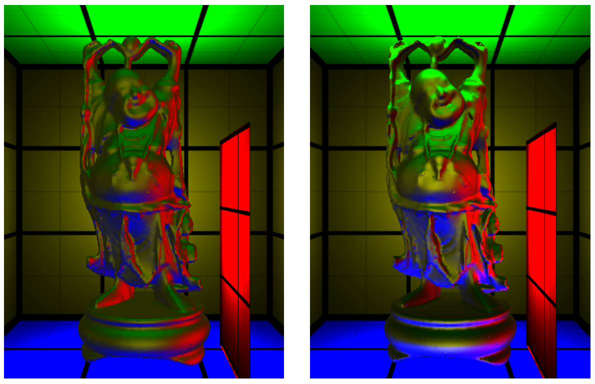


Figure 3: *Specular buddha rendered with the classical environment mapping method (left) and a reference image (right). The reference point for the environment mapping is located in the center of the room. Although the buddha’s head is expected to be greenish due to the green ceiling, the classical environment mapping is unable to meet this expectation.*

The issue of global computations to reduce the computation requirement of many lights has been addressed by *Precomputed Radiance Transfer* [Sloan et al. 2002; Kristensen et al. 2005], by methods aiming at ignoring possibly unimportant lights [Ward 1994; Shirley et al. 1996; Wald et al. 2003], and by approaches replacing a cluster of lights by a single point light source [Walter et al. 2005].

If only two-bounce indirect illumination is considered, we can use the shadow map instead of an environment map. As was pointed out in [Dachsbacher and Stamminger 2005], points where the light is bounced for the first time are stored by default in the shadow map. Adding illumination information to the depth values, the indirect illumination, i.e. the second bounce can be computed as the "reflection" of the shadow map.

The method proposed in this paper consider the last bounce of an arbitrarily evaluated coarse global illumination solution. The method works with cube maps, but does not require preprocessing to obtain them in sample points. In fact, thanks to the geometric localization procedure, it can use one cubemap for an object or even for several close objects. The cubemap is regenerated when the respective object moves significantly, but less frequently than the rendering frame rate of the application, amortizing he cubemap generation cost in multiple frames. Thus the new method can cope with fully dynamic scenes and changing illumination environments. The cubemap is downsampled, which corresponds to clustering lights and replacing a cluster by an area light source. Comparing to *Lightcuts* [Walter et al. 2005], our clustering is simpler, but is executed in real-time by the graphics hardware, and replaces the cluster by an area light. The radiance values of the single cube map are then localized for the shaded points taking into account the distances of the environment, the shaded points, and of the reference point, similarly to the approximate ray-tracing method [Szirmay-Kalos et al. 2005]. However, the new technique uses a different localization approach developed particularly for diffuse and glossy reflections.

3 The new algorithm

Let us assume that we use a single environment map that records illumination information for reference point $\vec{o}$. Our goal is to reuse this illumination information for other nearby points as well. To do so, we apply approximations that allow us to factor out those components from the rendering equation which strongly depend on shaded point $\vec{x}$.

In order to estimate the integral of the rendering equation, directional domain Ω' is partitioned to solid angles $\Delta\omega_i', i = 1, \ldots, N$, where the radiance is roughly uniform in each domain. After partitioning, the reflected radiance is expressed by the following sum:

$$L^r(\vec{x} \to \vec{\omega}) = \sum_{i=1}^{N} \int_{\Delta\omega_i'} L^{in}(\vec{x} \leftarrow \vec{y}(\vec{\omega}')) \cdot f_r(\vec{\omega}' \to \vec{x} \to \vec{\omega}) \cdot \cos\theta_{\vec{x}}\, d\omega'.$$

Let us consider a single term of this sum representing the radiance reflected from $\Delta\omega_i'$. If $\Delta\omega_i'$ is small, then L^{in} has small variation. Furthermore, considering the illumination and reflectance as random variables, the variation of illumination L^{in} depends on the surface of the *illuminating* points, while reflectance $f_r(\vec{\omega}' \to \vec{x} \to \vec{\omega}) \cdot \cos\theta_{\vec{x}}$ depends on the *shaded* point, thus they can be supposed to be independent. The expected value of the products of independent random variables (i.e. the integral of products) equals to the product of the expected values (i.e. the product of integrals), thus we can use the following approximation:

$$\int_{\Delta\omega_i'} L^{in}(\vec{x} \leftarrow \vec{y}(\vec{\omega}')) \cdot f_r(\vec{\omega}' \to \vec{x} \to \vec{\omega}) \cdot \cos\theta_{\vec{x}}\, d\omega' \approx$$

$$\approx \tilde{L}^{in}(\Delta y_i) \cdot \int_{\Delta\omega_i'} f_r(\vec{\omega}' \to \vec{x} \to \vec{\omega}) \cdot \cos\theta_{\vec{x}}\, d\omega' \qquad (1)$$

where $\tilde{L}^{in}(\Delta y_i)$ is the *average incoming radiance* coming from surface Δy_i seen at solid angle $\Delta\omega_i'$. Expressing average incoming radiance $\tilde{L}^{in}(\Delta y_i)$ on surface area Δy_i, we obtain:

$$\tilde{L}^{in}(\Delta y_i) = \frac{1}{\Delta y_i} \cdot \int_{\Delta y_i} L(\vec{y} \to \vec{\omega}_{\vec{y} \to \vec{x}})\, dy.$$

Note that this average is independent of shaded point $\vec{x}$ if the environment is diffuse, and can be supposed to be approximately independent of the shaded point if the environment is moderately glossy.

The second factor of the product in equation 1 is the *reflectivity* integral, which is also expressed as the product of the average integrand and the size of the integration domain:

$$\int_{\Delta\omega_i'} f_r(\vec{\omega}' \to \vec{x} \to \vec{\omega}) \cdot \cos\theta_{\vec{x}}\, d\omega' = a(\Delta\omega_i' \to \vec{x} \to \vec{\omega}) \cdot \Delta\omega_i'$$

where $a(\Delta\omega_i' \to \vec{x} \to \vec{\omega})$ is the *average reflectivity* from solid angle $\Delta\omega_i'$. The average reflectivity can be either obtained using only one directional sample on the fly, or precomputed and stored in a texture addressed by the angle of direction and the solid angle where averaging happens.

Putting the results of the average incoming radiance and the reflectivity formula together, the reflected radiance can be approximately expressed as

$$L^r(\vec{x} \to \vec{\omega}) \approx \sum_{i=1}^{N} \tilde{L}^{in}(\Delta y_i) \cdot a(\Delta\omega_i' \to \vec{x} \to \vec{\omega}) \cdot \Delta\omega_i'. \qquad (2)$$

4 Reusing illumination information

As concluded in the previous section, average incoming radiance values $\tilde{L}^{in}(\Delta y_i)$ are independent of the shaded point in case of diffuse or moderately glossy environment, thus these values can potentially be reused for all shaded points. To exploit this idea, visible surface areas Δy_i need to be identified and their average radiances need to be computed first. These areas are found and the averaging is computed with the help of a cube map placed at reference point $\vec{o}$ in the vicinity of the shaded object.We render the scene from reference point $\vec{o}$ onto the six sides of a cube. In each pixel of these images we store the radiance of the visible point and also the distance from the reference point. The pixels of the cube map thus store the radiance and also encode the position of small indirect lights.

The small virtual lights are clustered into larger area light sources while averaging their radiance, which corresponds to downsampling the cube map. A pixel of the lower resolution cube map is computed as the average of the included higher resolution pixels. Note that both radiance and distance values are averaged, thus finally we have larger lights having the average radiance of the small lights and placed at their average position. The total area corresponding to a pixel of a lower resolution cube map will be elementary surface Δy_i, and its average radiance is stored in the texel.

Supposing that the edge size of the cube map is 2, the solid angle subtended by a texel, i.e. by the surface seen through a texel, can be expressed with the *disc to point form factor approximation* [Glassner 1995] as:

$$\Delta\omega_i' \approx \frac{4}{M^2|\vec{L}|^3 + 4/\pi},$$

where M is the resolution of a single cubemap face and $\vec{L}$ is a vector pointing from the center of the cubemap to the texel (figure 4).

According to equation 2 the reflected radiance at the reference point is:

$$L^r(\vec{o} \to \vec{\omega}) \approx \sum_{i=1}^{N} \tilde{L}^{in}(\Delta y_i) \cdot a(\Delta\omega_i' \to \vec{o} \to \vec{\omega}) \cdot \Delta\omega_i'.$$

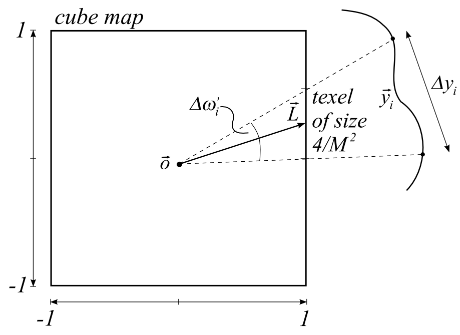


Figure 4: *Solid angle in which a surface seen through a cubemap pixel*

Let us now consider another point $\vec{x}$ close to the reference point $\vec{o}$ and evaluate a similar integral for point $\vec{x}$ while making exactly the same assumption on the surface radiance, i.e. it is constant in areas Δy_i:

$$L^r(\vec{x} \to \vec{\omega}) \approx \sum_{i=1}^{N} \tilde{L}^{in}(\Delta y_i) \cdot a(\Delta\omega_i' \to \vec{x} \to \vec{\omega}) \cdot \Delta\omega_i^*, \qquad (3)$$

where $\Delta\omega_i^*$ is the solid angle subtended by Δy_i from $\vec{x}$. Unfortunately, the solid angle values can be obtained directly from the geometry of the cubemap only if the shaded point is the center of the cube map. In case of other shaded points, special considerations are needed that are based on the distances from the environment surface.

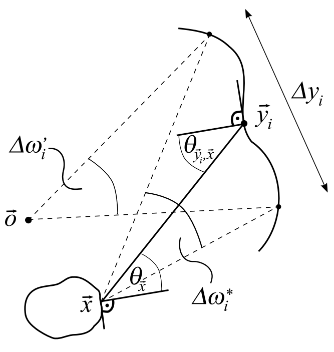


Figure 5: *The notations of the evaluation of subtended solid angles*

Solid angles $\Delta\omega_i'$ and $\Delta\omega_i^*$ can be expressed using the disc to point form factor approximation:

$$\Delta\omega_i' \approx \frac{\Delta y_i \cdot \cos\theta_{\vec{y}_i,\vec{o}}}{|\vec{o}-\vec{y}_i|^2 + \Delta y_i \cdot \cos\theta_{\vec{y}_i,\vec{o}}/\pi},$$

$$\Delta\omega_i^* \approx \frac{\Delta y_i \cdot \cos\theta_{\vec{y}_i,\vec{x}}}{|\vec{x}-\vec{y}_i|^2 + \Delta y_i \cdot \cos\theta_{\vec{y}_i,\vec{x}}/\pi}.$$

where $\theta_{\vec{y}_i,\vec{o}}$ and $\theta_{\vec{y}_i,\vec{x}}$ are the angle between the normal vector at $\vec{y}_i$ and the direction from $\vec{y}_i$ to $\vec{o}$, and the angle between the normal vector and the direction from $\vec{y}_i$ to $\vec{x}$, respectively (figure 5).

Assume that the environment surface is not very close compared to the distances of the reference and shaded points, thus the angles between the normal vector at $\vec{y}_i$ and reflection vectors from $\vec{o}$ and from $\vec{x}$ are similar ($\cos\theta_{\vec{y}_i,\vec{x}} \approx \cos\theta_{\vec{y}_i,\vec{o}}$). In this case, we can establish the following relationship between $\Delta\omega_i^*$ and $\Delta\omega_i'$:

$$\Delta\omega_i^* \approx \frac{|\vec{o}-\vec{y}_i|^2 \cdot \Delta\omega_i'}{|\vec{x}-\vec{y}_i|^2 \cdot (1-\Delta\omega_i'/\pi) + |\vec{o}-\vec{y}_i|^2 \cdot \Delta\omega_i'/\pi}.$$

5 Implementation

The proposed algorithm first computes an environment cube map from the reference point and stores the radiance and distance values of the points visible in its pixels. We usually generate $6 \times 256 \times 256$ pixel resolution cube maps. Then the cube map is downsampled to have M×M pixel resolution faces (M is 4 or even 2). Texels of the low-resolution cubemap represent elementary surfaces Δy_i whose average radiance and distance are stored in the texel. The illumination of these elementary surfaces is reused for an arbitrary point x, as shown by the following HLSL pixel shader program calculating the reflected radiance at this point:

```
half3 RefRad ( half3 N : TEXCOORD0, half3 V : TEXCOORD1,
               half3 x : TEXCOORD2 ) : COLOR0
{
  half3 Lr = 0;
  V = normalize( V );  N = normalize( N );
  for (int X = 0; X < M; X++) // for each texel
   for (int Y = 0; Y < M; Y++) {
     half2 t = half2((X+0.5f)/M, (Y+0.5f)/M);
     half2 l = 2 * t - 1;  // [0,1]->[-1,1]
     Lr += Cntr(x, half3(l.x,l.y, 1), N, V);
     Lr += Cntr(x, half3(l.x,l.y,-1), N, V);
     Lr += Cntr(x, half3(l.x, 1,l.y), N, V);
     // + similarly for the 3 remaining sides
  }
  return Lr;
}
```

The Cntr function calculates the contribution of a single texel of downsampled, low resolution cubemap LREnvMap to the illumination of the shaded point. Arguments x, L, N, and V are the relative position of the shaded point with respect to the reference point, the unnormalized illumination direction pointing to the center of the texel from the reference point, the unit surface normal at the shaded point, and the unit view direction, respectively. The following implementation obtains the average reflectivity on the fly taking one sample.

```
half3 Cntr(half3 x, half3 L, half3 N, half3 V) {
    half  l    = length(L);
    half  dw   = 4 / (M*M*l*l*l + 4/PI);
    half  doy  = texCUBE(LRCubeMap, L).a;
    half  doy2 = doy * doy;
    half3 y    = L / l * doy;
    half  dxy2 = dot(y-x, y-x);
    half  dws  = (doy2*dw) /
                 (dxy2*(1-dw/PI)+doy2*dw/PI);
    half3 I    = normalize(y - x);
    half3 H    = normalize(I + V);
    half3 a    = float3(0,0,0);
    if (dot(N,I)>0 && dot(N,V)>0)
       a = kd * max(dot(N,I),0) +
           ks * pow(max(dot(N,H),0),n);
    half3 Lin  = texCUBE(LRCubeMap, L).rgb;
    return Lin * a * dws;
}
```

First the solid angle subtended by the texel from the reference point is computed and stored in variable dw, then illuminating point y is obtained looking up the distance value of the cube map. The square distances between the reference point and the illuminating point, and between the shaded point and the illuminating point are put into `doy2` and `dxy2`, respectively. These square distances are used to calculate solid angle `dws` subtended by the illuminating surface from the shaded point. Phong-Blinn BRDF is used with diffuse reflectance `kd`, specular reflectance `ks`, and shininess `n`. Illumination direction `I` and halfway vector `H` are calculated, and the reflection of the radiance stored in the cube map texel is obtained according to equation 3.

6 Results

In order to demonstrate the results, we took a simple environment consisting of a cubic room with a divider face in it. The object to be indirectly illuminated is the bunny, happy buddha, and the the dragon, respectively. Each of these models consists of approximately 50-60 thousand triangles. Frame rates were measured in 700×700 windowed mode on an NV6800GT graphics card and P4/3GHz CPU. The first set of pictures (Figure 6) shows a diffuse bunny inside the cubic room. The images of the first column are rendered by the traditional environment mapping technique for diffuse materials where a precalculated convolution enables us to determine the irradiance at the reference point with a single lookup. Clearly, these precalculated values cannot deal with the position of the object, thus the bunny looks similar everywhere. The other columns show the results of our method using different sized cube maps. Note that even with extremely low resolution (2×2) we get images similar to the large-resolution reference.

The second set (Figure 7) and the third set (Figure 8) of pictures show a glossy buddha and a dragon inside a room. The first column presents the traditional environment mapping technique while the other three columns present the results of our localized algorithm. Similarly to the diffuse case, even cube map resolution of 2×2 produced more pleasing results than the classical technique. We have also implemented the proposed method in a game running at 30 FPS. Images of this game are shown by figure 9.

7 Conclusions

This paper presented a localization method for computing diffuse and glossy reflections of the incoming radiance stored in environment maps. The localization uses the distance values stored in the environment map texels. The presented method runs in real-time and provides visually pleasing results.

Acknowledgements

This work has been supported by OTKA (T042735) and GameTools FP6 (IST-2-004363) project.

References

BJORKE, K. 2004. Image-based lighting. In *GPU Gems,* R. Fernando, Ed. NVidia, 307–322.

BLINN, J. F., AND NEWELL, M. E. 1976. Texture and reflection in computer generated images. *Communications of the ACM 19*, 10, 542–547.

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games,* ACM Press, New York, NY, USA, 203–231.

DEBEVEC, P. 1998. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98*, 189–198.

G., G., SHIRLEY, P., HUBBARD, P., AND GREENBERG, D. 1998. The irradiance volume. *IEEE Computer Graphics and Applications 18*, 2, 32–43.

GLASSNER, A. 1995. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers, Inc., San Francisco.

GREENE, N. 1984. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications 6*, 11, 21–29.

KRISTENSEN, A., AKENINE-MOLLER, T., AND JENSEN, H. 2005. Precomputed local radiance transfer for real-time lighting design. In *SIGGRAPH 2005*.

MANTIUK, R., PATTANAIK, S., AND MYSZKOWSKI, K. 2002. Cube-map data structure for interactive global illumination computation in dynamic diffuse environments. In *International Conference on Computer Vision and Graphics*, 530–538.

RAMAMOORTHI, R., AND HANRAHAN, P. 2001. An efficient representation for irrandiance environment maps. *SIGGRAPH 2001*, 497–500.

REINHARD, E., TIJSSEN, L. U., AND JANSEN, W. 1994. Environment mapping for efficient sampling of the diffuse interreflection. In *Photorealistic Rendering Techniques*. Springer, 410–422.

SHIRLEY, P., WANG, C., AND ZIMMERMAN, K. 1996. Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics 15*, 1, 1–36.

SLOAN, P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH 2002 Proceedings*, 527–536.

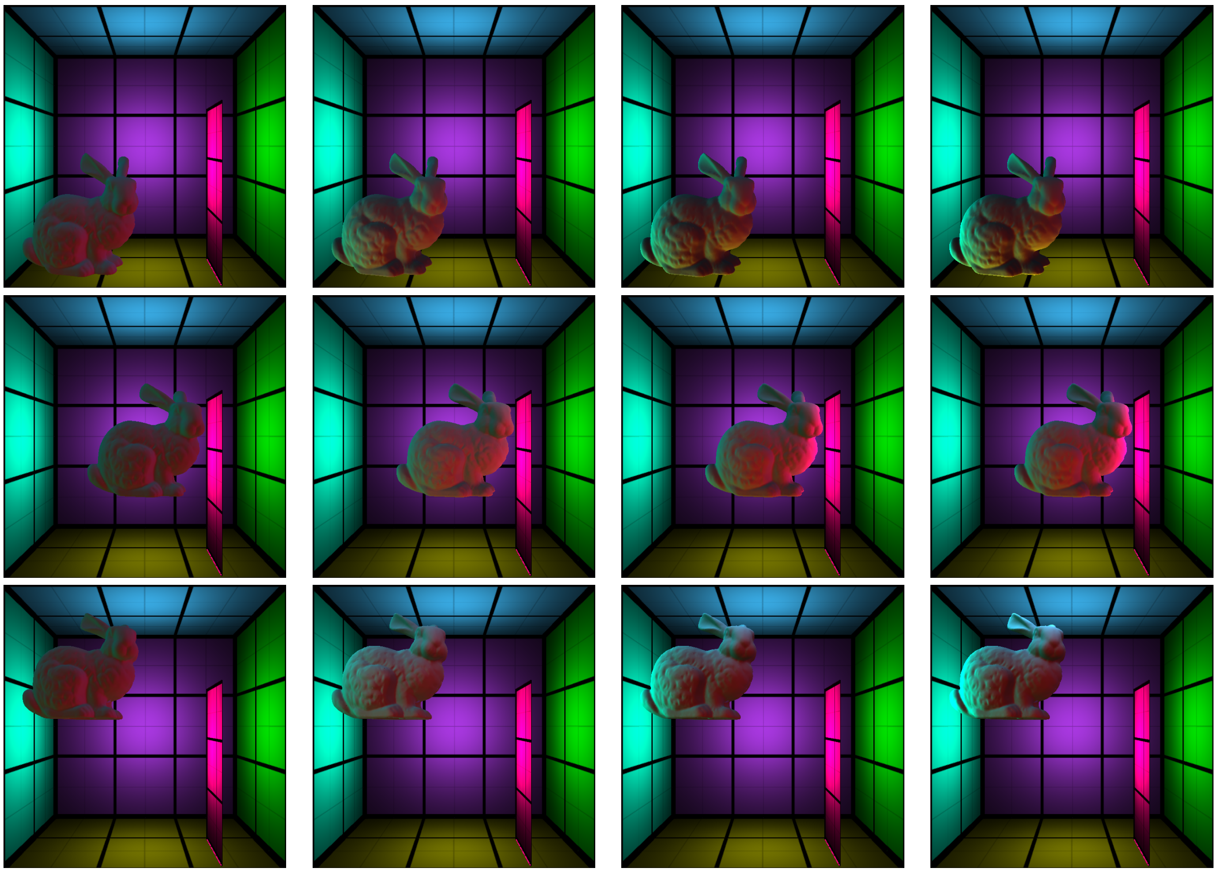
SZIRMAY-KALOS, L., ASZÓDI, B., LAZÁNYI, I., AND PREMECZ, M. 2005. Approximate ray-tracing on the GPU with distance impostors. *Computer Graphics Forum 24*, 3, 695–704.

WALD, I., BENTHIN, C., AND SLUSSALEK, P. 2003. Interactive global illumination in complex and highly occluded environments. In *14th Eurographics Symposium on Rendering*, 74–81.

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: A scalable approach to illumination. In *SIGGRAPH 2005*.

WARD, G. 1994. Adaptive shadow testing for ray tracing. In *Rendering Workshop '94*, 11–20.

ZHOU, K., HU, Y., LIN, S., GUO, B., AND SHUM, H.-Y. 2005. Precomputed shadow fields for dynamic scenes. In *SIGGRAPH 2005*.



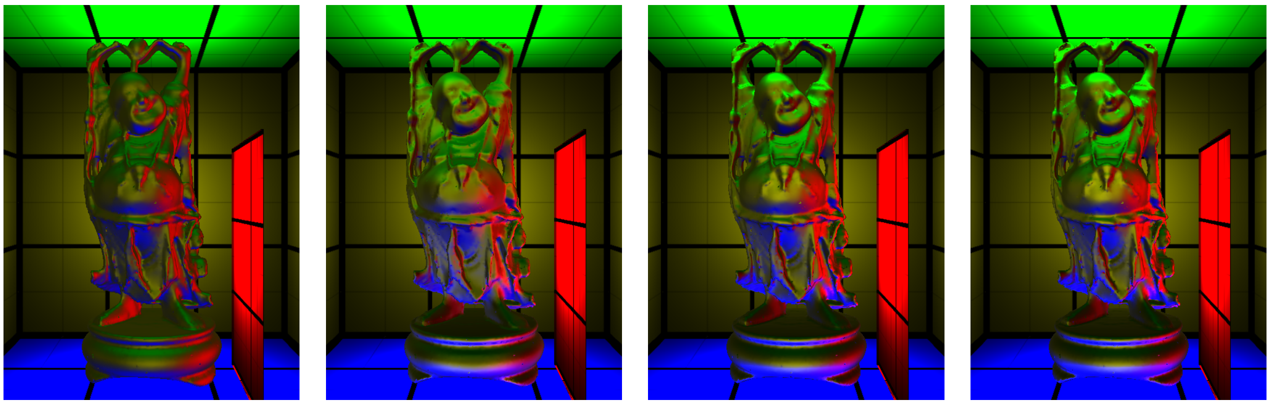
classical method
16 × 16 resolution

localized cubemap
2 × 2 resolution
75 FPS

localized cubemap
4 × 4 resolution
20 FPS

reference
16 × 16 resolution
2 FPS

Figure 6: *Diffuse bunny rendered with the classical environment mapping (left column) and with our algorithm using different environment map resolutions.*



classical method
16 × 16 resolution

localized cubemap
2 × 2 resolution
22 FPS

localized cubemap
4 × 4 resolution
6 FPS

reference
16 × 16 resolution
1 FPS

Figure 7: *Specular buddha (the shininess is 5) rendered with the classical environment mapping (left column) and with our algorithm using different environment map resolutions.*

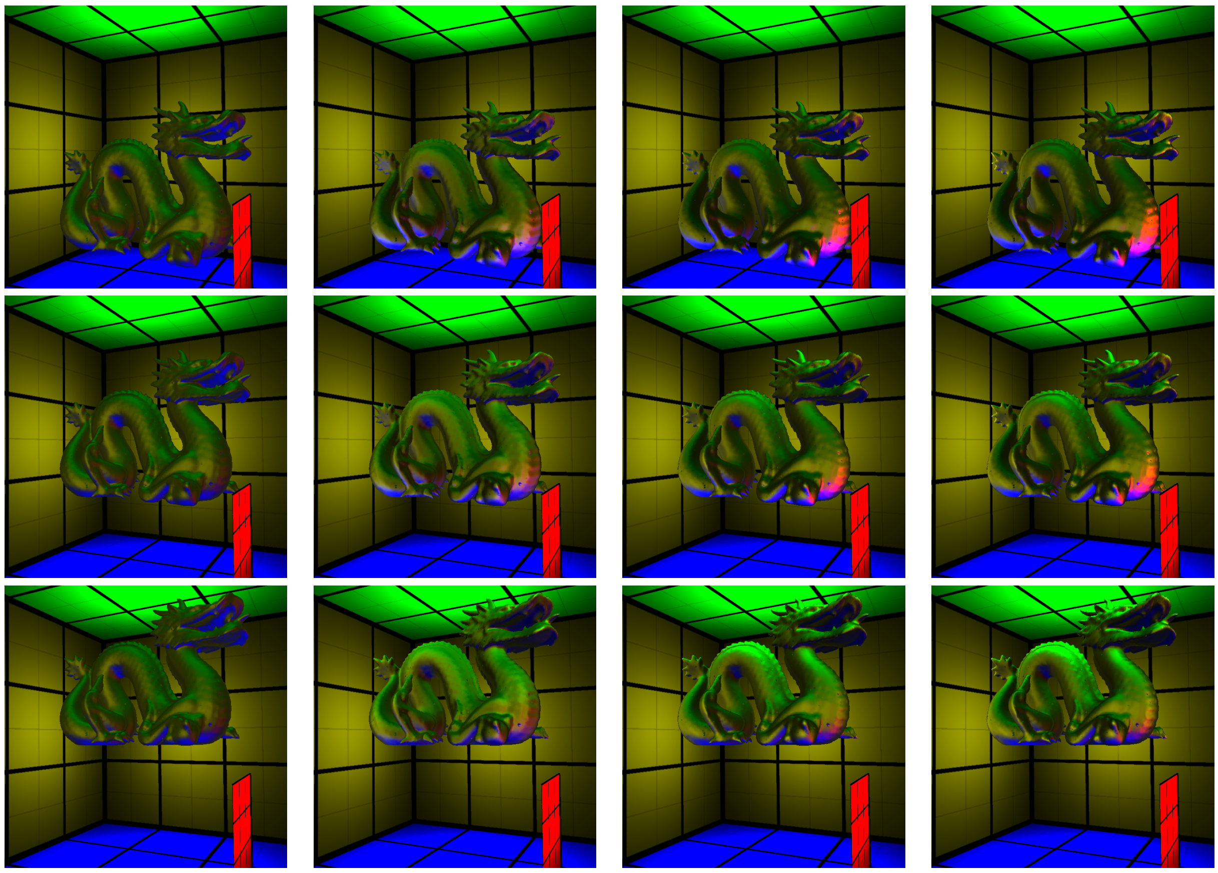


Figure 8: *Specular dragon (the shininess is 5) rendered with the classical environment mapping (left column) and with our algorithm using different environment map resolutions.*

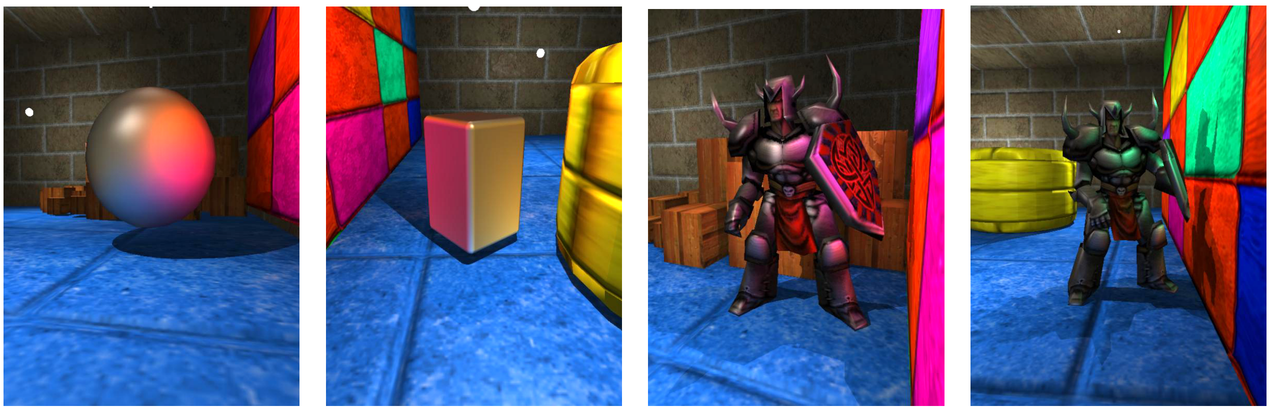


Figure 9: *Glossy objects in a game running at 30 FPS.*