Lighting Precomputation Using the Relighting Map

Tien-Tsin Wong (ttwong@acm.org)
The Chinese University of Hong Kong

Chi-Sing Leung (eeleungc@cityu.edu.hk)
City University of Hong Kong

Kwok-Hung Choy
City University of Hong Kong

Introduction

Lighting is a key component in creating dramatic atmosphere in game scenes, especially in horror games. However, it is always difficult to achieve dynamic lighting due to the scene complexity, intensive lighting computation, and the scarce computing resource during the runtime. Techniques for lighting precomputation reduce the runtime workload by shifting the major computation to an offline phase. One simple and popular example is to store the precomputed luminosity information in the lightmap (texture) and blend it with the surface texture to produce lighting effect. Unfortunately, such lighting is static.

It would be nice to have a variable lightmap that changes dynamically according to the light source. To do so, we need a way to compactly represent hundreds or even thousands of lightmaps, and a way to retrieve the desired lightmaps in real-time. In this article, we propose a representation, *the relighting map*, which can be used to represent variable lightmap. It stems from the previous work [Wong97a] in image-based relighting. With such representation, dynamic lighting can be achieved as follows. In the preprocessing phase, the game developer first prepares arbitrary number of representative lightmaps using any modeler and renderer. These precomputed reference lightmaps are then encoded into a set of highly compressed relighting maps. Each relighting map is simply a texture. During the runtime, the lightmap corresponding to the desired lighting condition can be synthesized (relit) in real-time by linearly combining these relighting maps. For those games (such as *Resident Evil* and *Alone in the Dark*) that have a fixed viewpoint, their backgrounds can be represented by the relighting map as a whole.

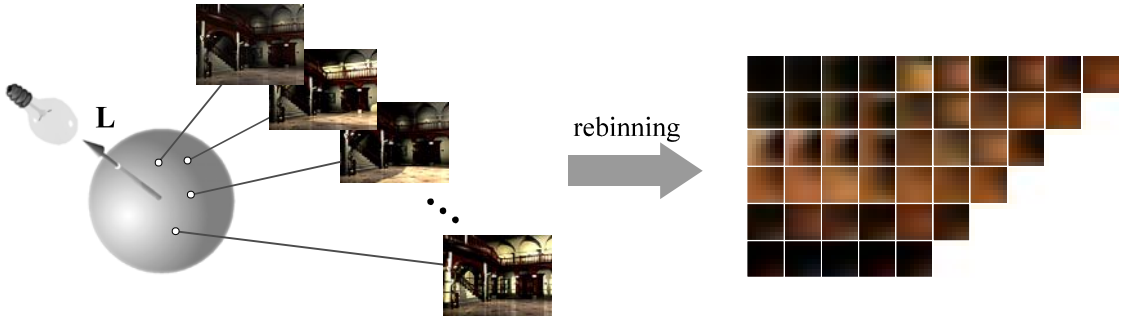
The relighting map technique employs a locally supported *spherical radial basis function* instead of the globally supported spherical harmonics used in previous work [Wong97a][Wong03]. In other words, it is more capable in capturing high-frequency lighting features such as shadow and specular highlight. Moreover, the homogeneity and simplicity of spherical radial basis function makes the real-time rendering more efficient and practical than that of spherical harmonics.

Preparing the Relighting Maps

Input Images

The reference images (or lightmaps) are created with a directional light source as the sole illuminant. Each of them corresponds to a distinct light vector (**L**) sampled on a unit

sphere, as illustrated on the left hand side of Figure 1. There can be arbitrary number of samples scattered on the sphere. The only requirement is the light vector must be known. The massive color values are then rebinned in such a way that all values related to the same pixel window are grouped together. The right hand side of Figure 1 illustrates the rebinning. Each tile corresponds to one such group of color values. It tells us the color of that pixel when the scene is illuminated by a varying directional light. Note that the smoothness of color in a tile facilitates the compression.



[INSERT FIGURE 1 HERE]

Figure 1. Reference images are rebinned to maximize data correlation for compression.

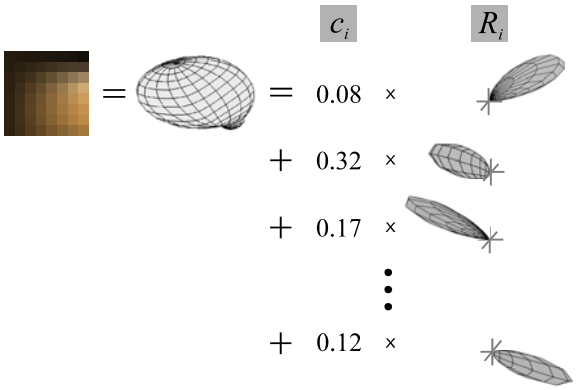
Each tile is in fact a spherical function, as every color value inside corresponds to a sample scattered on sphere. We call it the *Apparent BRDF* (ABRDF) [Wong97a]. Interestingly, it is closely related to Bidirectional Reflectance Distribution Function (BRDF). It differs from BRDF in that it may contain high-frequency shadow and other global illumination effects, such as caustics. Such high-frequency lighting effects will be over-blurred by the globally supported spherical harmonic which is commonly used for BRDF representation.

Spherical Radial Basis Function

Instead, we propose to use the spherical radial basis function [Broomhead88][Jenison96]. Its locality nature is desirable in capturing shadows, highlights and caustics. The basic idea is to approximate each ABRDF by a linear combination of k spherical radial basis functions R_i,

$$\sum_{i=0}^{k-1} c_i R_i(\mathbf{L}) \qquad (1)$$

Figure 2 illustrates such approximation graphically. Intuitively speaking, the ABRDF is approximated by a weighted sum of bumps (spherical radial basis functions R_i), each oriented in different direction. The input to R_i is the light vector $\mathbf{L}$ that actually looks up the value on sphere. The basis function R_i returns scalar value. The more radial basis functions are employed, the more accurate the approximation is. A useful number of basis functions k is 15 to 35. Since R_i are predefined and common for all ABRDFs, only the k coefficients c_i have to be stored for representing each ABRDF.



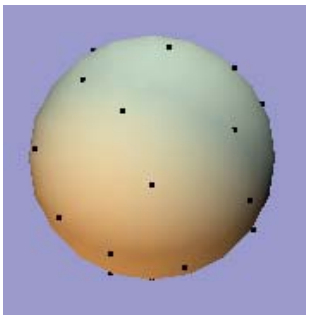
[INSERT FIGURE 2 HERE]

Figure 2. Color values associated with the same pixel window are approximated by the linear combination of differently oriented spherical radial basis function.

Each spherical radial basis function (Equation 2) is actually a Gaussian function of the angle between the light vector **L** and the corresponding radial basis center $\mathbf{Q}_i$ laid on sphere. Both **L** and $\mathbf{Q}_i$ must be normalized. The center $\mathbf{Q}_i$ determines the orientation of the bump. All bumps have the same spread Δ. The spread Δ is selected as the minimal geodesic distance between two neighboring centers. The centers $\mathbf{Q}_i$ and the spread Δ are pre-selected before the encoding process.

$$R_i(\mathbf{L}) = \exp\left(-\frac{\left[\cos^{-1}(\mathbf{L}\cdot\mathbf{Q}_i)\right]^2}{2\Delta^2}\right) \tag{2}$$

To represent the radiance in different direction, centers $\mathbf{Q}_i$ are uniformly distributed on sphere. There are several methods for generating uniformly distributed points on sphere. In particular, we use a deterministic method to generate a stochastic-like pattern, known as Hammersley point set (Figure 3). The source code for generating the Hammersley point set is available on the homepage listed in [Wong97b].

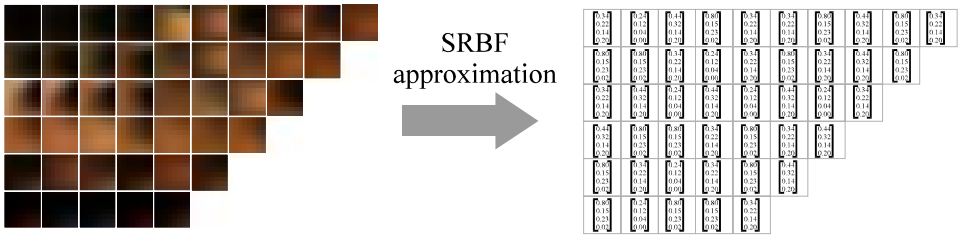


[INSERT FIGURE 3 HERE]

Figure 3. The distribution of 30 radial basis centers generated using Hammersley point set.

The Relighting Maps

In other words, after the approximation, the array of tiles in Figure 1 is converted to an array of *k*-dimensional coefficient vectors, as shown in Figure 4. Let's call these vectors the SRBF vectors from now on, where SRBF stands for spherical radial basis function. The total size of the SRBF vectors is much smaller than that of the original color values. Hence, compression is achieved.

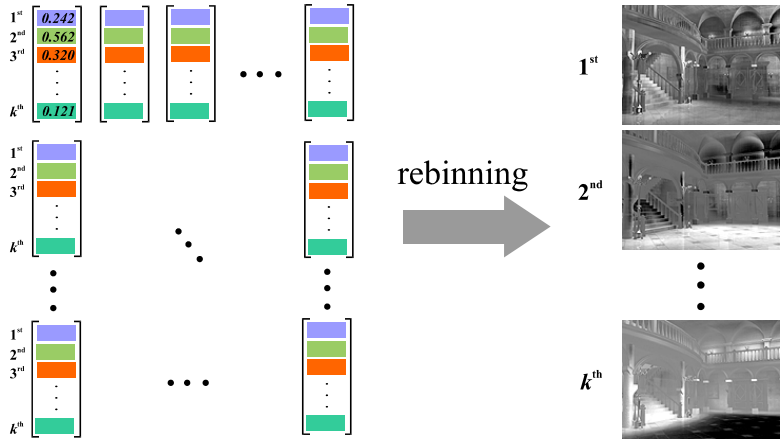


[INSERT FIGURE 4 HERE]

Figure 4. The spherical radial basis function approximation converts the ABRDF's (tiles on the left) to SRBF vectors for storage reduction.

Equation 1 is the key to relighting. Given a specific **L**, Equation 1 allows us to quickly look up (reconstruct) a pixel value due to that lighting direction without reconstructing the whole ABRDF. To relight the whole image, Equation 1 is performed for all pixels. Obviously, this computation is fully *parallelizable*. Hence it allows us to use SIMD-based GPU to achieve real-time rendering.

In order to parallelize the computation, we need to rebin the array of *k*-dimensional SRBF vectors to form *k relighting maps*, as in Figure 5. They are rebinned in the following manner. The first coefficients of all SRBF vectors are grouped to form the first relighting map. This process is repeatedly applied to form other relighting maps. Each relighting map is an image of real values (can be positive or negative). These relighting maps represent the precomputed lighting effect in the form of textures. They are the data to be stored on disk and they will be loaded as textures for rendering. By linearly combining these maps, we can simulate different lighting effects.



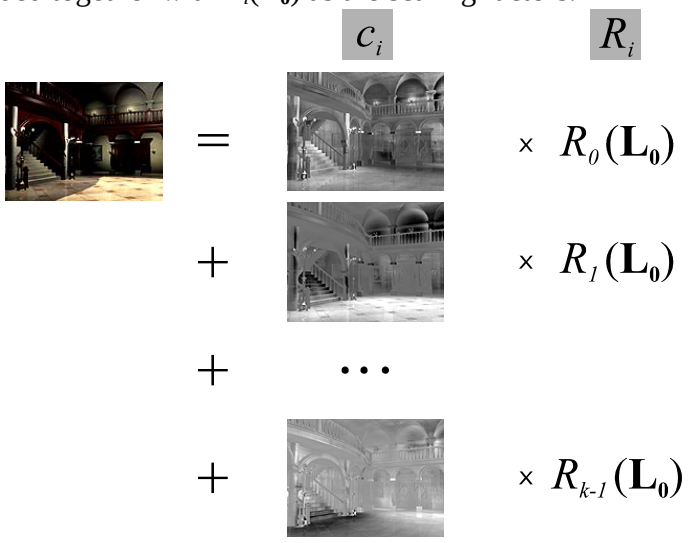
[INSERT FIGURE 5 HERE]

Figure 5. The array of *k*-dimensional SRBF vectors are rebinned to form *k* relighting maps.

Directional-Source Relighting

Image-wise Linear Combination

With these relighting maps, we can synthesize complex lighting effects by image-wise or pixel-wise linear combination of them. The rendering process is actually a reconstruction process that independently computes Equation 1 for every pixel in the image/lightmap. The computation reconstructs (or "looks up") a color value from the encoded SRBF coefficients by feeding a light vector **L**. The simplest case of relighting is directional-source relighting. In this case, every pixel are fed with the same light vector $\mathbf{L_0}$ and hence the same $R_i(\mathbf{L_0})$. Therefore, the relighting can be regarded as a linear combination of relighting maps c_i and scalars $R_i(\mathbf{L_0})$ in Figure 6. The evaluation of $R_i(\mathbf{L_0})$ can simply be done by software instead of GPU, as there are only *k* evaluations. Then, the relighting maps can be blended together with $R_i(\mathbf{L_0})$ as the scaling factors.



[INSERT FIGURE 6 HERE]

Figure 6. Relighting by a directional light is a simple image-wise linear combination of relighting maps and weights $R_i(\mathbf{L_0})$.

Directional-Source Shader

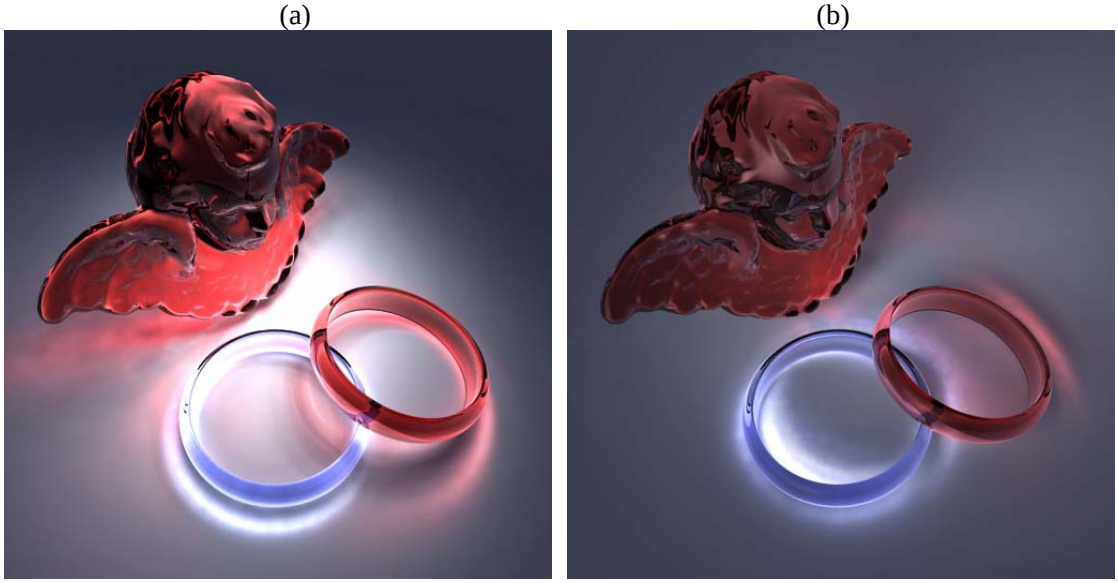
We use OpenGL and Cg to implement a directional-source relighting shader. First of all, the relighting maps are loaded into the texture units of GPU. Since the number of textures allowed for processing in one shader pass is usually limited, the relighting process has to be divided into multiple passes. The intermediate result should be stored in an offline pixel buffer for accumulation. The following code shows the linear combination of 3 relighting maps.

```
float4 DirShader(
  uniform samplerRECT c0,                    // map c0
  uniform samplerRECT c1,                    // map c1
  uniform samplerRECT c2,                    // map c2
  uniform float R0,                          // R0(Lo)
  uniform float R1,                          // R1(Lo)
  uniform float R2,                          // R2(Lo)
  uniform float4 lightcolor,                 // light color
  float3  texCoord : TEXCOORD0
): COLOR
{
  float4 acc;

  acc   = h4texRECT(c0,texCoord.xy) * R0;    // c0R0
  acc  += h4texRECT(c1,texCoord.xy) * R1;    // c1R1
  acc  += h4texRECT(c2,texCoord.xy) * R2;    // c2R2
  acc  *= lightcolor;
  acc.w = 1;
  return acc;
}
```

Note that R_i returns positive value while the relighting maps may contain negative values. Depending on available resource, the relighting maps can be stored as standard 8-bit integer textures or floating-point textures (16-bit half float is usually sufficient). Even with 8-bit precision, the image quality may not be reduced too much. If integer textures are used, attention must be paid on handling the numeric range of values. These relighting maps can be further compressed in memory using hardware supported texture compression such as S3TC.

Figure 7 shows the directional-source relighting of two scenes, “house” and “caustics”. Scene “house” in Figures 7 (a) & (b) mimics typical horror game scene. The lighting effect can be easily modified in real-time to achieve the desired horror atmosphere. Twenty relighting maps are used in this example. Scene “caustics” in Figures 7 (c) & (d) demonstrates the ability of spherical radial basis function in capturing rapidly changing caustics and shadow. Thank to the locality nature of spherical radial basis function, only 25 relighting maps are used to achieve the real-time relighting. Note that it takes 40 min. to render one frame in “caustics” data set on Pentium IV 2.8 GHz CPU using Mental Ray.

(a) (b)



(c) (d)

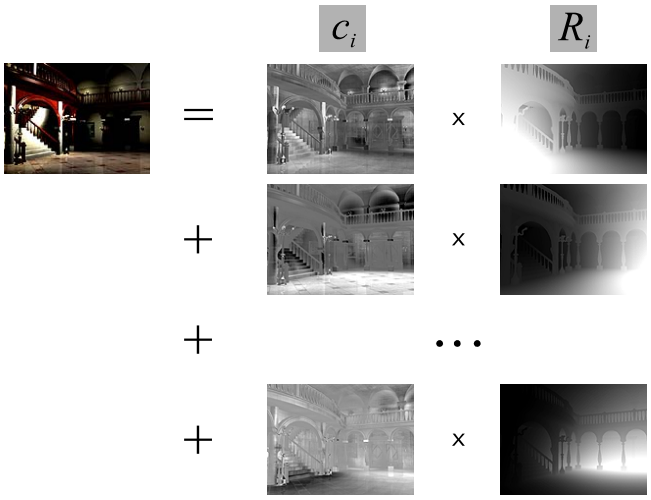
[INSERT FIGURE 7 HERE]

Figure 7. Two scenes relit by directional light source. (a) and (b) show the “house” scene while (c) and (d) show the “caustics” scene.

Point-Source Relighting

Per-pixel Linear Combination

Point-source relighting is basically the same as that of directional light source. The major difference is that the light vector **L** is different for each pixel. Hence, $R_i(\mathbf{L})$ have to be evaluated for each pixel. Figure 8 illustrates such difference when compared to Figure 6. $R_i(\mathbf{L})$ are maps of scalars instead of scalars. Instead of a linear combination of images as in Figure 6, we now have a per-pixel linear combination of color values. Note that the scalar maps $R_i(\mathbf{L})$ look like “partially illuminated” depth maps.



[INSERT FIGURE 8 HERE]

Figure 8. Relighting by a point light source requires a per-pixel linear combination of color values.

Computing the Light Vector

The major difficulty of point-source relighting is the computation of R_i. To do so, we have to first compute the light vector for each pixel. Given the depth map, the light vector is computed by the following equation,

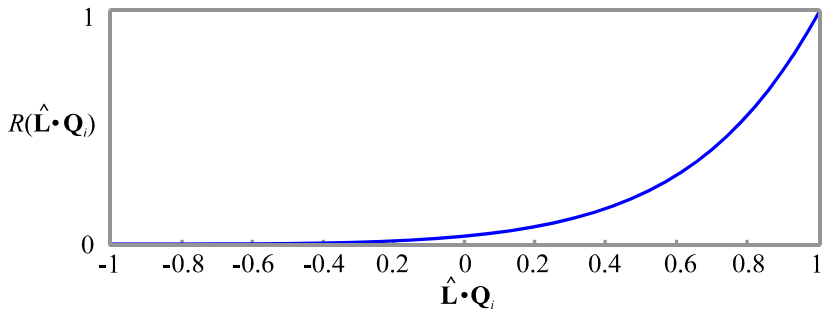
$$\mathbf{L} = \mathbf{S} - \left(\mathbf{E} - \frac{\mathbf{V}}{|\mathbf{V}|} d \right) \tag{3}$$

where **S** is the position of the point light source; **E** is the viewpoint; **V** is the viewing direction associated with the interested pixel; and d is the depth value of that pixel. Since **E**, **V** and d are all known and only **S** varies during the runtime, we can precompute $\mathbf{P} = \mathbf{E} - d\,\mathbf{V}/|\mathbf{V}|$ (the intersection points between the viewing rays **V** and the scene) and store them as a vector map. During the runtime, the equation **L** = **S** – **P** is computed to determine the light vector **L** in a per-pixel manner.

Evaluating the Basis Functions

One major advantage of spherical radial basis function over spherical harmonic basis functions is its *homogeneity*. In the case of spherical harmonic basis functions, different basis functions are different and higher order ones are more complex than the lower order ones. The *heterogeneity* of spherical harmonics imposes difficulty in the real-time evaluation of spherical harmonic basis functions (there can be millions of evaluations). In [Wong03], the evaluation of the spherical harmonic basis functions is achieved by looking up a series of precomputed cubemaps. However, the limited resolution of cubemaps introduces error, especially for high-order basis functions. As the number of basis functions increases, the number of cubemaps also has to be increased. Hence this approach becomes impractical when a large number of spherical harmonic basis functions are used.

On the other hand, all spherical radial basis functions have the same Guassian form except with different centers $\mathbf{Q}_i$ (Equation 2). Therefore, the computations needed in evaluating different basis functions are the same. In practice, the evaluation of R_i in Equation 2, can be easily accomplished by looking up a high-resolution *1D table* with $\hat{\mathbf{L}} \cdot \mathbf{Q}_i$ as the index, where $\hat{\mathbf{L}}$ denotes the normalized **L**. This table is precomputed to store the evaluations of $R(\hat{\mathbf{L}} \cdot \mathbf{Q}_i)$. The domain of R_i is [-1,1] while its range is [0,1]. Figure 9 plots this function. Hence, each evaluation requires only a dot product, $\hat{\mathbf{L}} \cdot \mathbf{Q}_i$, and a table lookup.

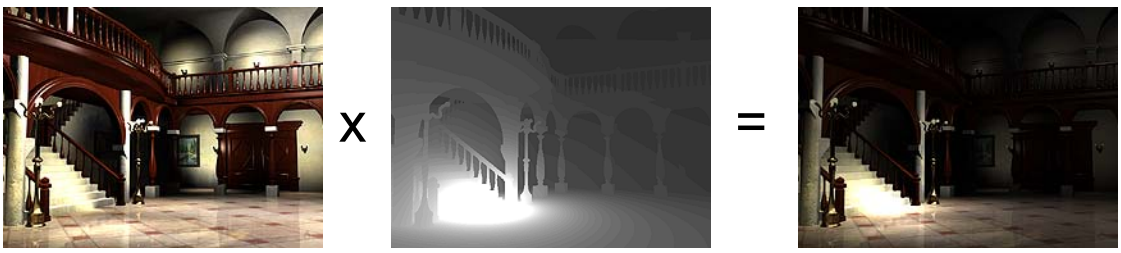


[INSERT FIGURE 9 HERE]
Figure 9. The evaluation of R_i can be done by a dot product and a lookup of this 1D table.

Figure 9 shows that R_i increases as the light vector **L** approaches $\mathbf{Q}_i$ and decreases rapidly as **L** leaves $\mathbf{Q}_i$. Hence it explains the bump shape of R_i in Figure 2. As the light vector **L** is derived from the depth map and the Gaussian fall-off surrounds $\mathbf{Q}_i$, it explains why the R_i maps in Figure 8 look like "partially-illuminated" depth maps. The direction $\mathbf{Q}_i$ actually determines which part of the image is "illuminated".

Attenuation

To model the distance fall-off effect of a point light source, we can simply multiply the linearly combined result by an attenuation factor. Figure 10 illustrates the attenuation graphically. The attenuation factor is obtained from the light vector (without normalization). The attenuation formula we use is $a_o / |\mathbf{L}|$, where a_o is a user-defined constant; and $|\mathbf{L}|$ is the magnitude of **L** or the distance from the light source to the intersection point.



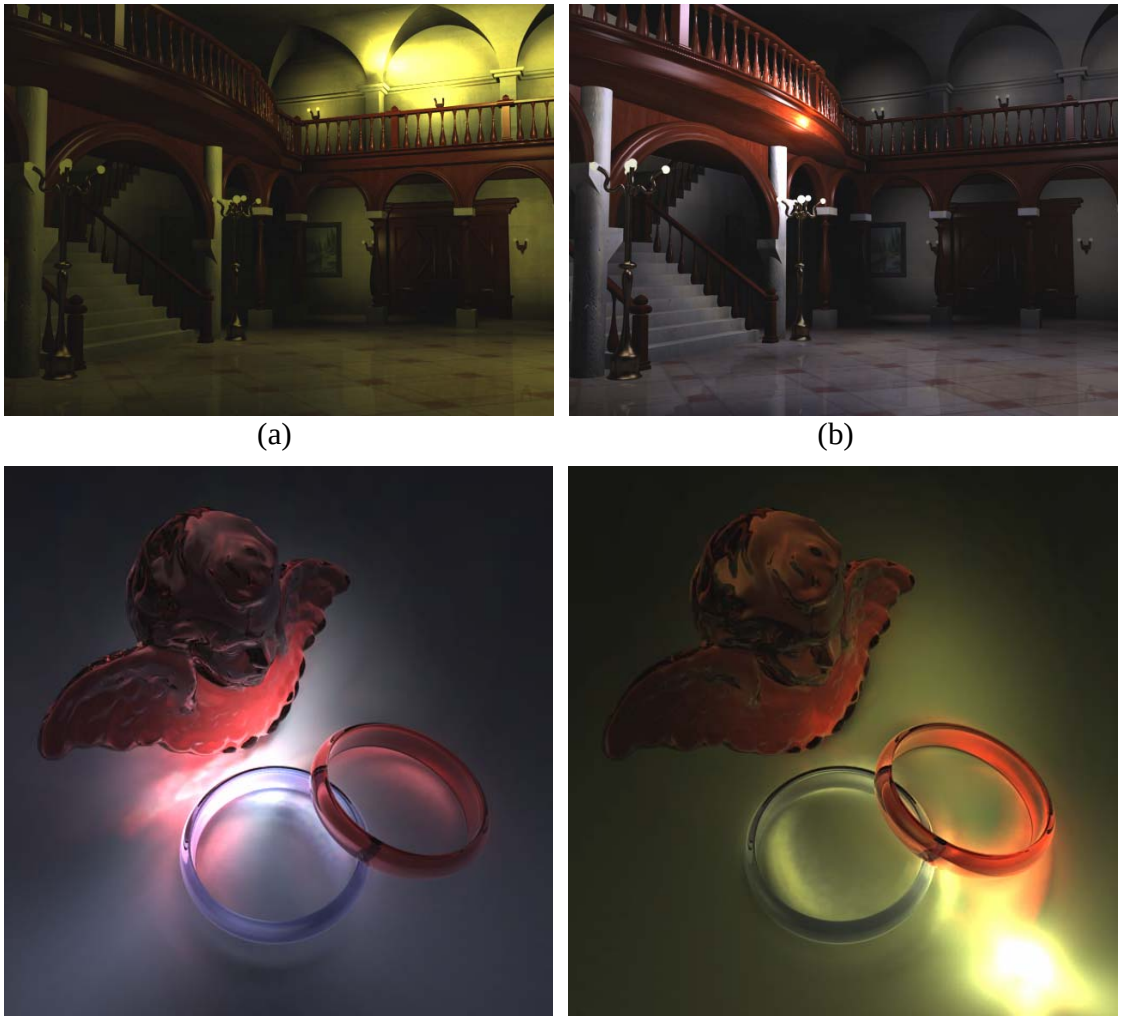
[INSERT FIGURE 10 HERE]
Figure 10. The distance fall-off effect is incorporated by multiplying the linearly combined result with an attenuation factor.

Point-Source Shader

The following point-source relighting shader is implemented in Cg. It computes the light vector **L** from the depth map, computes the dot product $\hat{\mathbf{L}} \cdot \mathbf{Q}_i$, evaluates R_i by table lookup, and applies the attenuation. For program clarity, only three relighting maps are combined in this code.

```
float4 PtShader(
  uniform samplerRECT c0,                    // map C0
  uniform samplerRECT c1,                    // map C1
  uniform samplerRECT c2,                    // map C2
  uniform float3      center0,               // Q0
  uniform float3      center1,               // Q1
  uniform float3      center2,               // Q2
  uniform samplerRECT ipmap,                 // map of intersection points
  uniform samplerRECT RBFtable,              // R()
  uniform float       htablesize,            // half size of R table
  uniform float3      lightpos,              // S, position of light source
  uniform float4      lightcolor,            // light color
  uniform float       a_o,                   // ao, attenuation scaling factor
  float3  texCoord : TEXCOORD0
): COLOR
{
  float4  acc;                               // accumulation variable
  float   atten;                             // attenuation
  half3   P;                                 // P, point of intersection
  float3  L;                                 // L, light vector
  float2  lookup;                            // lookup index

  P        = h3texRECT(ipmap,texCoord.xy);   // lookup P
  L        = (lightpos - P);                 // L = S - P
  atten    = a_o / length(L);                // attenuation = ao/|L|
  L        = normalize(L);                   // normalize L
  lookup.y = 0.5;

  // Evaluate and accumulate c0R0
  lookup.x = (dot(L,center0)+1) * htablesize; // L·Q0
  acc      = h1texRECT(RBFtable,lookup)*h4texRECT(c0,texCoord.xy);

  // Evaluate and accumulate c1R1
  lookup.x = (dot(L,center1)+1) * htablesize; // L·Q1
  acc     += h1texRECT(RBFtable,lookup)*h4texRECT(c1,texCoord.xy);

  // Evaluate and accumulate c2R2
  lookup.x = (dot(L,center2)+1) * htablesize; // L·Q2
  acc     += h1texRECT(RBFtable,lookup)*h4texRECT(c2,texCoord.xy);

  acc     *= atten*lightcolor;
  acc.w    = 1;
  return acc;
}
```

Figure 11 shows the point-source relighting of two scenes, “house” and “caustics”. With the relighting maps, we can approximate the complex “caustics” effect in real-time on a normal PC. Note the change of caustics near the rings and angel in Figures 11(c) and (d).



(a) (b)

(c) (d)

[INSERT FIGURE 11 HERE]

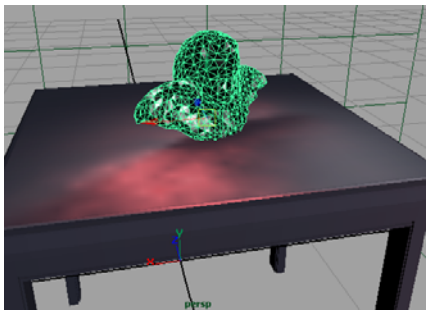
Figure 11. Two scenes relit by point light sources. (a) and (b): “house” scene relit by yellow and white point sources respectively. (c) and (d): “caustics” scene relit by white and yellow point sources respectively.

The Variable Lightmap

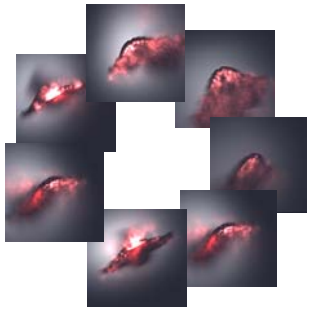
We now illustrate an example usage of the relighting map as a variable lightmap in representing time-consuming lighting effects, caustics and shadow. In this example, we model the caustics and shadow cast by the semi-transparent angel statue on a table. Both the viewpoint and light source move in real-time. We first model a 3D scene using a modeler (Figure 12(a)). Then, we generate a set of precomputed lightmaps (Figure 12(b)) and encode them as the relighting maps. These lightmaps are actually the photon maps on the table.

During the runtime, the rendering is divided into multiple passes. In the first pass, the current lightmap is synthesized from the relighting maps according to the current light

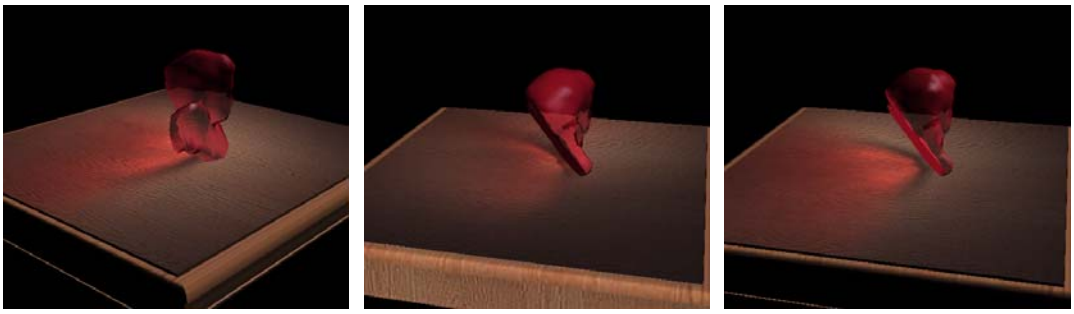
source position. Then an image is generated from the current viewpoint with the generated lightmap mounted on the table. Note that the semi-transparent angel is not rendered at this moment. Finally, the angel is rendered with the previously generated image as background to simulate the refraction. The rendered angel is then superimposed onto the previously generated image to return the final image. With this approach, the time-consuming caustics computation can be shifted to offline, with the expense of some simple linear combination of relighting maps. Figure 12 (c) shows the screenshots from the real-time demo (see the companion CDROM).



(a) Design the 3D scene



(b) Generate the lightmaps



(c) Screen shots from the real-time caustics demo

[INSERT FIGURE 12 HERE]

Figure 12. The variable lightmap.

Conclusion

In this article, we have illustrated how to represent the expensive precomputed lighting effects using spherical radial basis function. The coefficients of spherical radial basis functions are organized to form the relighting maps. The synthesis of lighting effect is basically a linear combination of relighting maps either in an image-wise (directional light source) or a pixel-wise (point light source) manner. The locality nature of relighting map captures shadows, highlights, and caustics. Its homogeneity and simplicity offers a practical and simple solution to point source and spotlight rendering, which is tedious in spherical harmonic base. It also offers a scalable solution. Depending on the amount of available resource on GPU, the developer can trade for higher image quality by keeping more relighting maps, or less resource consumption by lowering the image quality.

Acknowledgements

We would also like to thank Jianqing Wang for preparing the “caustics” and “lightmap” data sets, Kin-Ting Lam for preparing the “house” data set, Ka-Ling Fok, Lai-Sze Ng, and Ping-Man Lam for preparing the demos. The work is supported by the Research Grants Council of the Hong Kong Special Administrative Region, under RGC Earmarked Grants (Project No. CUHK 4189/03E and CityU 1122/01E).

References

[Broomhead88] Broomhead, D. S. and D. Lowe, “Multivariate Functional Interpolation and Adaptive Networks,” *Complex Systems*, Vol. 2, pp. 321-355, 1988.

[Jenison96] Jenison, R. L. and K. Fissell, “A Spherical Basis Function Neural Network for Modeling Auditory Space,” *Neural Computation*, Vol. 8, pp. 115-128, 1996.

[Wong97a] Wong, Tien-Tsin, et al., “Image-based Rendering with Controllable Illumination,” Proceedings of the 8-th Eurographics Workshop on Rendering, St. Etienne, France, June 1997, pp 13-22.

[Wong97b] Wong, Tien-Tsin, et al., “Sampling with Hammersley and Halton Points,” *Journal of Graphics Tools*, Vol. 2, No. 2, 1997, pp. 9-24. http://www.cse.cuhk.edu.hk/~ttwong/papers/udpoint/udpoints.html

[Wong03] Wong, Tien-Tsin, et al., “Real-Time Relighting of Compressed Panoramas,” *Graphics Programming Methods*, Charles Rivers Media, 2003, pp. 375-288.