Photon Map Gathering on the GPU†

Szabolcs Czuczor, László Szirmay-Kalos, László Szécsi and László Neumann

Budapest University of Technology and Economics and University of Girona

Abstract

Photon mapping methods obtain the indirect illumination of a point by finding those photon hits that arrived at the neighborhood of the point on the object surface. This paper proposes a method that stores the photon hits in a texture of the graphics hardware and replaces the traditional kd-tree based neighborhood searches by the filtering of this texture. This step finds the irradiance of all points (i.e. all texels) simultaneously in a single step, thus the average irradiance of a point can be obtained by a single texture lookup. Using this approach we can port the final gathering step of photon mapping to the graphics hardware (GPU). The CPU is only responsible for generating new light paths and updating the unfiltered photon map. Thanks to the optimal subdivision of the computation work between the the CPU and the GPU, the proposed algorithm can render globally illuminated scenes interactively.

1. Introduction

Global illumination is a physical simulation, which includes both direct and indirect lighting. While direct lighting can be computed by the GPU, the indirect lighting is often computed on the CPU, for example, by photon mapping. Photon mapping generates photon hits on the surfaces of the virtual scene. If we put the photon hits into a texture (which later can be mapped onto the surfaces), the implementation of the indirect illumination computation is possible by programmable pixel and vertex shaders.

2. Photon mapping

Photon mapping is a two-phase global illumination algorithm [7, 6, 8]. In the first phase a lot of light paths are generated originating at the light sources and bouncing at the surfaces randomly. The random generation of the paths is usually governed by BRDF sampling and Russian roulette. At each surface hit the Monte Carlo estimation of the incoming power is computed and stored. The data structure representing these hits is called the *photon-map*. The photon-map is usually organized in a *kd-tree* to support efficient photon retrieval. During this retrieval process we need those photons that are in the neighborhood of the point of interest. A photon hit is stored with the power of the photon on different wavelengths, position, direction of arrival, and with the surface normal.

The photon map represents the indirect illumination, which can be taken into account when the reflected radiance of a point is obtained. This calculation is called *final gathering*. Suppose we need to determine reflected radiance L of point $\vec{x}$ in direction ω. The gathering phase is based on the following approximation of the light transport operator:

$$L(\vec{x},\omega) = \int_{\Omega'} L^{in}(\vec{x},\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta' \, d\omega' \approx$$

$$\sum_{i=1}^{n} \frac{\Delta\Phi(\vec{x}_i,\omega_i')}{\Delta A} \cdot f_r(\omega_i',\vec{x},\omega), \qquad (1)$$

where L^{in} is the incoming radiance, f_r is the BRDF, θ' is the angle between the surface normal and the incoming direction, and $\Delta\Phi(\vec{x}_i,\omega_i')$ is the power of a photon landing at $\vec{x}_i$ of surface ΔA from direction ω_i'. The $\Delta\Phi$ and ΔA quantities are approximated from the photons in the neighborhood of $\vec{x}$ in the following way. A sphere centered around $\vec{x}$ is extended until it contains n photons. If at this point the radius of the sphere is r, then the intersected surface area is $\Delta A = \pi r^2$.

This original version of the photon-map algorithm has

† This work has been supported by the GameTools FP6-004363 EU project, OTKA ref. No.: T042735., Spanish-Hungarian Join Action, E-26/04

several drawbacks, including the large storage space needed for the photon hits, the time consuming final gathering and the filtering artifacts caused by using the n photons nearby. In order to reduce these artifacts, we should take into account only those photon hits that arrived at the same surface, or more precisely, when the surface normal associated with the photon hit is similar to that of the shaded point. On the other hand, the approximation of the area from where the photons are gathered by an intersection circle can be very inaccurate at surface boundaries. Here a more precise area calculation is needed as suggested by [5]. Christensen [2] proposed another improvement to compute diffuse interreflections. In the preprocessing phase, the irradiance is estimated at each photon hit from the other photon hits that are nearby. Thus during final gathering, we do not have to find the n nearest photons, but only the closest one where the normal vector is similar to the normal vector of the given point.

3. Photon tracing with improved density estimation

Algorithms like photon mapping can also be discussed as tools to solve a density estimation problem [9]. Photon hits represent a sampling of the irradiance, from which a smooth reconstruction of the reflected radiance should be generated. To achieve this, a convolution operation executing low pass filtering is defined:

$$L(\vec{x},\omega) \approx \sum_{i=1}^{n} \Delta\Phi(\vec{x}_i,\omega_i') \cdot f_r(\omega_i',\vec{x},\omega) \cdot k(\vec{x}_i - \vec{x}), \qquad (2)$$

where $\vec{x}_i$ is the location of the ith photon hit and k is the filter kernel, which can be integrated to 1.

The density estimation can be implemented both with a gathering [7] or a shooting approach [10]. In case of shooting, the photon hits are splat as small textured quads onto the object surfaces and alpha blending is used to add up their contribution.

4. Computation of the reflected radiance by texture filtering

In our approach, the photon hits are stored in texture space [1]. Since neighborhood searches are also executed in the texture space, we look for a neighborhood of the respective texel. Usually if two points belong to the same surface and are close, their texels will also be close. Thus when we gather photon hits from the neighboring texels, we can assume that these hits would be close to the shaded point in object space as well. However, it can happen that two 3D points are far while their respective texels are close. Obviously in this case we cannot use the photon hits of the other point for the radiance calculation. To recognize these cases, the surface identities (id) should also be stored in the texture. When the neighborhood is built and if the ids of neighboring texels are different, the neighborhood must not be extended with those texels. On the other hand, we should also store the surface normals to avoid considering those photon hits that have significantly different orientation.

Having established an appropriate neighborhood in texture space, the surface area corresponding to this neighborhood is computed as required by equation 1. This computation can again be supported by information stored in texture. During preprocessing the value of the surface area corresponding to a given texel is computed and stored. The texel areas are calculated as the ratio of the area of the surface elements (triangles) in world space and in texture space. When the neighborhood of a texel is built, these area values are summed.

Summarizing, our method handles the steps of photon mapping as follows: (I.) Photon tracing on CPU. (II.) Photon hit conversion into textures. (III.) Final gathering step in two passes: (III/1.) filtering the view dependent reflected radiance at those points which correspond to texel centers (see section 5 for details), and (III/2.) mapping this texture on the object surface during a normal, local illumination rendering.

5. Computation of the reflected radiance

The reflected radiance computation (the photon map filtering method) is implemented as a pixel shader program on the GPU. Our proposed algorithm works with the following textures: `diffusemap` (the diffuse reflectivities), `photonhit` (the original photon hit powers), `photondir` (the incoming direction of the hits), `surface` (the surface id and the area of the surface in channel r and g respectively), `normmap` (the x, y and z coordinates of surface normals in the channels r, g and b respectively).

The pixel shader of the radiance computation checks texels in the $(2N+1) \times (2N+1)$ square neighborhood of sample point of texture coordinate `pIN.uv`, and decides whether they could be added to the neighborhood. Using variable filter kernels, we need a texture (`filter`) storing the filter values in the $(2N+1) \times (2N+1)$ texel neighborhood. We cannot assume that the kernel integrates to one, because not necessarily all texels will be taken into account due to different surfaces and normal vectors. To handle this problem, a normalization constant is calculated similarly to the `area`. For texels belonging to the neighborhood, their hit power is multiplied with the BRDF of the point of interest and the resulting reflected radiance is added to the reflected radiance of the point. The BRDF function takes incoming direction of the photon `indir` and eye direction `pIN.eye`, which is computed by the vertex shader and is interpolated by the graphics hardware before calling this pixel shader code.

The presented program works with textures storing the normal vector, the surface id, and surface area of those points they are associated with. These textures can be generated using render-to-texture technology. Figure 1 shows the normal and area maps of one of our test scenes.

The pixel shader program of our algorithm is the following:

```
float3 surf0 = tex2d(surface, pIN.uv);
float3 norm0 = tex2d(normmap, pIN.uv);
float area = 0;
for(int dy = -N; dy <= N; dy += 1)
  for(int dx = -N; dx <= N; dx += 1) {
    float2 uv1 = pIN.uv + float2(dx, dy);
    float  k = tex2d(filter, uv1);
    float2 surf = tex2d(surface, uv1);
    float3 norm = tex2d(normmap, uv1);
    if(surf.r == surf0.r &&
       dot(norm, norm0) > threshold ) {
      area += surf.g * k;
      float3 pow = tex2d(photonhit, uv1) * k;
      float3 indir = tex2d(photondir, uv1);
      rad += pow * BRDF(indir,pIN.uv,pIN.eye);
    }
  }
}
return rad/area;
```

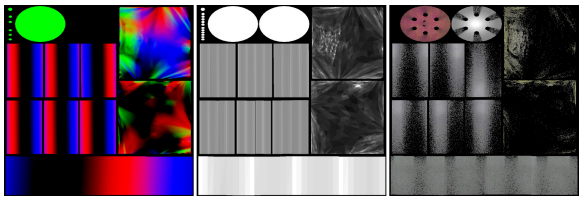

Figure 1: *The normal (left), the area (middle) and the photon map of the armadillo scene*

To speed up and optimize the algorithm we can take another approach to implement the filtering mechanism. We do not calculate filtered photon map texels in a pixel shader program using a lot of texture look-ups in loops, but we draw the unwrapped photon map many times on itself with predefined offsets (corresponding to the kernel elements) using alpha blending. See Figure 2.

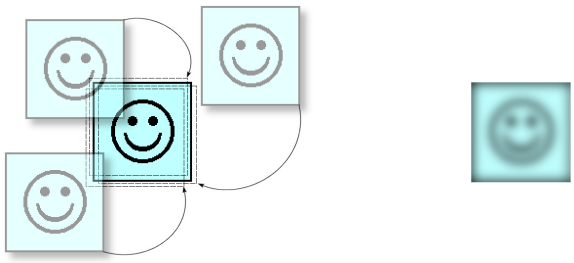

Figure 2: *Filtering by alpha blending. On the left the original texture is blended many times onto itself offseted and weighted corresponding to the position and value of the kernel elements. On the right the resulting map can be seen.*

6. Generating the photon textures

Having generated all photon hits by simulating random walks starting at the light sources on CPU and computed the texture coordinates of these hits, they are written into the maps of photon powers and incoming directions and uploaded to the GPU. Photon directions are not needed if the surfaces are diffuse, but play a significant role in case of specular materials. Note that it can happen that two or more photon hits fall on the same texel, when the photon hits are very dense at this surface region. The probability of this event grows as we decrease the resolution of the photon map texture. This is not a problem for diffuse reflections, since their BRDF is independent of the incoming direction, thus powers can be added. However, the direction dependent specular reflections cannot be handled in this way, but we should purge photon hits and reduce their density where they are highly concentrated. This operation should maintain the total power arriving at this surface region, i.e. when photons are ignored, the power of the kept photons must be scaled up.

In order to investigate this problem mathematically, let us revisit equation 2 expressing the reflected radiance, and group the terms according to the texels:

$$L(\vec{x},\omega) \approx \sum_{i=1}^{n} \Delta\Phi(\vec{x}_i,\omega_i') \cdot f_r(\omega_i',\vec{x},\omega) \cdot k(\vec{x}_i-\vec{x}) =$$

$$\sum_{j=1}^{N}\sum_{i=1}^{n_j} \Delta\Phi(\vec{x}_j,\omega_{ji}') \cdot f_r(\omega_{ji}',\vec{x},\omega) \cdot k(\vec{x}_j-\vec{x}),$$

where texel j is expected to store photon hits $\Delta\Phi(\vec{x}_j,\omega_{ji}')$ for all $i = 1,\ldots,n_j$.

In case of diffuse reflections, the sum belonging to a single texel is:

$$\sum_{i=1}^{n_j} \Delta\Phi(\vec{x}_j,\omega_{ji}') \cdot f_r(\vec{x}) \cdot k(\vec{x}_j-\vec{x}) =$$

$$f_r(\vec{x}) \cdot k(\vec{x}_j-\vec{x}) \cdot \sum_{i=1}^{n_j} \Delta\Phi(\vec{x}_j,\omega_{ji}'),$$

thus the powers of the photon hits can be simply added.

For specular reflections, let us approximate the sum belonging to a single texel applying Monte Carlo methods. We find a probability density p_{ji} so that $\sum_{i=1}^{n_j} p_{ji} = 1$, sample an integer i^* with this density, and approximate the sum as:

$$\sum_{i=1}^{n_j} \Delta\Phi(\vec{x}_j,\omega_{ji}') \cdot f_r(\omega_{ji}',\vec{x},\omega) \cdot k(\vec{x}_j-\vec{x}) \approx$$

$$k(\vec{x}_j-\vec{x}) \cdot \frac{\Delta\Phi(\vec{x}_j,\omega_{ji^*}') \cdot f_r(\omega_{ji^*}',\vec{x},\omega)}{p_{ji^*}}.$$

In order to make the variance of this random estimation

small, selection probabilities are set proportional to the luminance of the photon hits. Let us denote the luminance of wavelength dependent power Φ by $\mathcal{L}(\Phi)$. The probability of selecting photon hit i is:

$$p_{ji} = \frac{\mathcal{L}(\Phi(\vec{x}_j, \omega'_{ji}))}{\sum_{l=1}^{n_j} \mathcal{L}(\Phi(\vec{x}_j, \omega'_{jl}))}.$$

Thus, having sampled hit i^*, the Monte Carlo estimate of the reflected radiance is:

$$k(\vec{x}_j - \vec{x}) \cdot \frac{\Delta\Phi(\vec{x}_j, \omega'_{ji^*})}{\mathcal{L}(\Phi(\vec{x}_j, \omega'_{ji}))} \cdot f_r(\omega'_{ji^*}, \vec{x}, \omega) \cdot \sum_{l=1}^{n_j} \mathcal{L}(\Phi(\vec{x}_j, \omega'_{jl})).$$

The Monte Carlo estimate introduces a small variance in the estimation. However, filtering reduces this variance and makes the variance of neighboring pixels correlated, thus we can still avoid noise artifacts in the image.

7. Discussion and Conclusions

We used two test scenes (Figure 3). Initially two million photons were emitted. The armadillo scene$^{(1)}$ contains 3335, the Cornell box scene$^{(2)}$ contains 144 facets. The computer configuration was: WinXP, AMD AthlonXP 1492 MHz CPU, NVIDIA GeForce 6800 GT graphics card. The photon maps and the area map is generated off-line on CPU, the normal map is created on GPU only once before the normal workflow. For details see Table 1.

The proposed approach can be used for interactive walkthrough animations and even in interactive global illumination. In the latter case even photon maps should be regenerated between frames. In order to cope with the speed requirements, we do not rebuild all photon paths in each frame, just those that have been changed with high probability. These photon paths are identified by selective photon tracing [3].

	Photon map	**Filter kernel**		
resolution	512×512	11×11	7×7	3×3
FPS$^{(1)/(2)}$		**0.8 / 1.6**	**2.0 / 3.8**	**11 / 20**
resolution	256×256	11×11	7×7	3×3
FPS$^{(1)/(2)}$		**1.8 / 3.6**	**4.4 / 8.9**	**22 / 45**

Table 1: *Speed and performance results.*

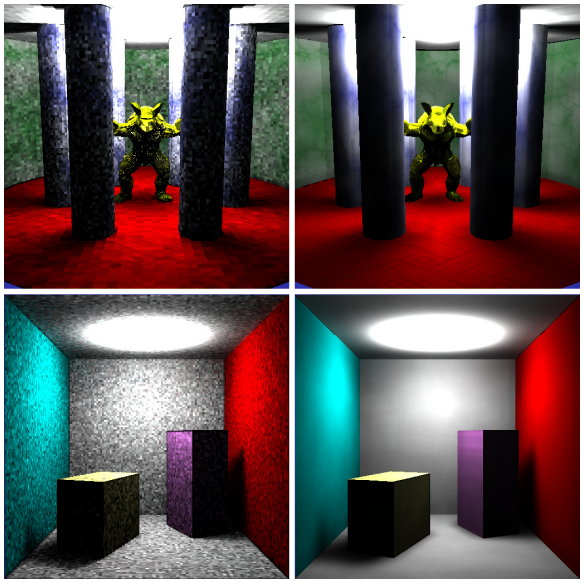

Figure 3: *The armadillo and Cornell box scene (with unfiltered and filtered photon map). Photon map resolution:* 256×256*, kernel size:* $7 \times 7^{(1)}$ *and* $11 \times 11^{(2)}$

References

[1] R. Bastos, M. Goslin, and H. Zhang. Efficient radiosity rendering using textures and bicubic reconstruction. In *ACM-SIGGRAPH Symposium on Interactive 3D Graphics*, 1997. 2

[2] P. Christensen. Faster photon map global illumination. *Journal of Graphics Tools*, 4(3):1–10, 2000. 2

[3] K. Dmitriev, S. Brabec, K. Myszkowski, and H-P. Seidel. Interactive global illumination using selective photon tracing. In *Rendering Techniques 2002*, 2002. 4

[4] T. Hachisuka. Final gathering on GPU. In *ACM Workshop on General Purpose Computing on Graphics Processors*, 2004.

[5] Heinrich Hey and Werner Purgathofer. Advanced radiance estimation for photon map global illumination. *Computer Graphics Forum (Proceedings of Eurographics 2002)*, 21(3), September 2002. 2

[6] H. W. Jensen. Global illumination using photon maps. In *Rendering Techniques '96*, pages 21–30, 1996. 1

[7] H. W. Jensen and N. J. Christensen. Photon maps in bidirectional Monte Carlo ray tracing of complex objects. *Computers and Graphics*, 19(2):215–224, 1995. 1, 2

[8] H. W. Jensen and P. H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. *Computers and Graphics (SIGGRAPH '98 Proceedings)*, pages 311–320, 1998. 1

[9] P. Shirley, B. Wade, P. Hubbard, and D Zareski. Global illumination via density-estimation radiosity. In *Eurographics Rendering Workshop '95*, 1995. 2

[10] Wolfgang Sturzlinger and Rui Bastos. Interactive rendering of globally illuminated glossy scenes. In *Rendering Techniques '97 (8th EG Workshop on Rendering)*, pages 93–102, 1997. 2