Scalable photon splatting for global illumination

Fabien Lavignotte, Mathias Paulin

Scalable Photon Splatting for Global Illumination

Fabien Lavignotte Mathias Paulin

IRIT University Paul Sabatier
Toulouse France

Abstract

In this paper, we present a new image based method for computing efficiently global illumination using graphics hardware. We propose a two pass method to compute global lighting at each pixel. In the first pass, photons are traced from the light sources and their hit points are stored. Then, in the second pass, each photons hit point is splatted on the image to reconstruct the irradiance. The main advantages of our method in comparison with previous approaches is scalability. Indeed, it can be used to obtain coarse solution as well as accurate solution and is well suited for complex scenes. First, we present the density estimation framework that is the theory behind our reconstruction pass to understand how to control the accuracy with our method. Then, we show how to use graphics hardware to improve reconstruction time with some restrictions but without losing the ability to compute an accurate simulation.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity;

Keywords: global illumination, density estimation, graphics hardware, photon tracing

1 Introduction

The goal of global illumination is to resolve the light transport integral [Kajiya 1986] to generate a photorealistic image from a scene description. Algorithms based on unbiased Monte Carlo methods have been proposed to resolve the light transport integral. They consist in sampling the transport paths from the light sources to the camera lens. Clever sampling of the light transport paths allow to represent various complex lighting effects such as caustics, soft shadows, glossy reflections and indirect diffuse lighting [Veach and Guibas 1995] [Veach and Guibas 1997]. But these methods are time consuming and images are still noisy even after sampling a large number of light transport paths because of the slow convergence of Monte Carlo methods.

Multi-pass methods resolve the lack of efficiency in pure Monte Carlo techniques. Some parts of the light transport integral are computed using biased but more efficient method, as radiosity to take into account diffuse indirect lighting [Chen et al. 1991] or particle tracing to render caustics on planar surfaces using texture [Heckbert 1990]. The purpose of multi-pass methods is to present a good balance between slow and accurate algorithm and efficient but biased method. Nevertheless, the biased method must be consistent.

Currently, the most popular multi-pass method is the photon map [Jensen 1996]. First, a photon tracing pass generates a set of photons hit points stored in a balanced kd-tree. In fact, two sets, called photon map, are generated : a global photon map to reconstruct indirect lighting, and a high density caustic photon map to reconstruct caustics. Then, a Monte Carlo ray tracer is used to compute direct lighting and reflections. Radiance is reconstructed from the photon map using nearest neighbor density estimation, nearest photons being efficiently located using the kd-tree.

The drawback of photon map is that the set of photons hit points must be stored in main memory during the second pass, so the accuracy of the photon map is limited. A method has been proposed to control the storage of the photons [Suykens and Willems 2000], avoiding the store too many photons in region of high density. This is still a drawback for the global photon map since a large number of photons is needed. So, to overcome this problem, the photon map estimate for indirect lighting is used only after a first bounce or gathering step, but the final gathering step must be highly sampled to avoid noisy artifacts. To be more efficient, some methods avoid to recompute the final gathering step at each pixel [Ward and Heckbert 1992] [Christensen 1999]. The error of these methods is quite difficult to control.

In this paper, we propose a new technique to reconstruct diffuse and caustic lighting effects using a two pass method quite similar to photon mapping. First, we simulate light transport by a photon tracing pass, and then lighting is reconstructed using an image based method that takes advantages of current graphics hardware. Our method is based on photon splatting method [Sturzlinger and Bastos 1997]. This preliminary approach had problems with accuracy of reconstruction and was limited to simple scene. So, our goal is that photon splatting becomes a useful tool in global illumination so we try to resolve these problems in this paper. First, we present the theory behind our algorithm : the density estimation framework [Walter 1998]. Then, we show how to implement our method on graphics hardware without losing accuracy even with a large number of photons hit points. Also, we present how to reconstruct accurately discontinuities features of the lighting such as caustics using adaptive reconstruction. Last, some implementation details and practical applications of our algorithm are given.

2 Density estimation framework

Density estimation framework is composed of two basic phases : a photon tracing phase and a reconstruction phase based on density estimation. The density estimation framework is related to many different methods : highly accurate view independent solution [Walter 1998], indirect lighting textures [Myszkowski 1997], caustics only textures [Heckbert 1990] [Granier et al. 2000], and photon map [Jensen 1996].

2.1 Photon tracing

Photon tracing consists in tracing random walks or paths from the light sources. The name of photons is a bit misleading because

actually our 'photons' does not match any physical meaning. A photon is a vertex of a random walk, it consists of a position x, a direction w and an associated weight ϕ or 'power'.

The photons are generated from the light sources according to a probability density function $p_e(x,\omega,\lambda)$ where x is the initial position of the photon, ω its initial direction and λ its wavelength. The power ϕ carried by a photon for a particular wavelength is then :

$$\phi = \frac{L_e(x,\omega,\lambda)\cos\theta}{N p_e(x,\omega,\lambda)} \tag{1}$$

where L_e is the self-emitted radiance, θ is the angle between the normal of the surface at x and the direction ω, and N is the number of photons traced from the light sources. Then the photon is traced in the scene and when a photon hit a surface, its hit point y is stored for the next phase. At y, the photon is either absorbed or reflected according to the BRDF using russian roulette. If reflected the new power ϕ' carried by the photon is :

$$\phi' = \frac{f(y,\omega',\lambda)\cos\theta'}{p_r(y,\omega',\lambda)}\phi$$

where f is the BRDF, ω is the incoming direction, ω' the new direction of the photon chosen with the probability density function $p_r(y,\omega',\lambda)$ and θ' the angle between ω and ω'.

Probability density function are chosen in order to give the same importance for each photon. That means each photon carry the same power.

$$p_e(x,\omega,\lambda) \sim L_e(x,\omega,\lambda)\cos\theta$$
$$p_r(y,\omega',\lambda) = f(y,\omega,\lambda)\cos\theta'$$

In the MGF standard, the self-emitted radiance is defined as constant in position and direction for each geometric primitive. With this assumption, p_e is defined as follow. First, the wavelength is chosen according to a pdf $p(\lambda)$, a light is selected using the pdf $\frac{\Phi_L(\lambda)}{\Phi_T(\lambda)}$. Then the point x is selected uniformly on the light using pdf $\frac{1}{A_L}$, and finally the direction ω is chosen according to the pdf $\frac{\cos\theta}{\pi}$. As a result using (1), the power ϕ for a particular wavelength is simplified to:

$$\phi = \frac{\Phi_T(\lambda)L_e(\lambda)A_L\pi}{N_p p(\lambda)\Phi_L(\lambda)} = \frac{\Phi_T(\lambda)}{N_p p(\lambda)} \tag{2}$$

2.2 Density Estimation

Once the photon tracing phase is completed, the irradiance is reconstructed using the stored photon hit points. This is equivalent to a density estimation problem as shown first by Heckbert [Heckbert 1990]. Indeed, the photon hit points are distributed accorded to a probability density function $p(x,\lambda)$ that is proportional to the irradiance $E(x,\lambda)$:

$$p(x,\lambda) = \frac{E(x,\lambda)}{\int_S E(y,\lambda)dy} \tag{3}$$

The integration of the irradiance in the environment can be replaced by $n\phi$ with n the total number of photons and ϕ the power carried by each photon for a specific wavelength. Because this pdf is unknown, a non parametric density estimation is used: the kernel density estimation [Silverman 1986].

The kernel density estimator constructs an estimate $\hat{p}$ of an unknown pdf p from a set of n observed data points $X_1,\ldots,X_n$:

$$\hat{p}(x) = \frac{1}{nh^d}\sum_{i=1}^{n} K(\frac{x-X_i}{h}) \tag{4}$$

where d is the dimension of x, $K(x)$ is a kernel function with unit area, generally symmetric and equal to zero if $|x| > 1$ and h is the bandwidth or smoothing parameter.

One of the main interesting properties of the kernel density estimator is the trade-off between variance and bias provided by the bandwidth h. Indeed, the integrated bias is proportional to h^2 and the integrated variance is proportional to $(nh)^{-1}$ [Silverman 1986].

So, for our problem, (3) and (4) are combined to give the following estimator for computing the irradiance of a surface point x:

$$E(x,\lambda) = \frac{\phi}{h^2}\sum_{i=1}^{n} K(\frac{x-X_i}{h}) \tag{5}$$

3 Photon splatting

3.1 Principles

Our purpose is to reconstruct the irradiance at each pixels efficiently using the estimator (5).The contribution of any photon hit point X to the estimator at surface point x is:

$$c(X) = \frac{\phi}{h^2}K(\frac{x-X}{h}) \tag{6}$$

The kernel function $K(x)$ in the estimator is equal to zero if $|x| > 1$, that means that the contribution of a photon hit point X to the estimator in x is null if the distance between x and X is above the bandwidth h. So, the irradiance estimator can be evaluated by two different techniques :

- For each pixels, find the photon hit points X which distance from the visible point x is below h and sums their contribution.
- For each photon hit points X add its contribution to the estimator of each pixels which distance from X is below h

when traced from a light source,

The key observation is that the second technique corresponds to splatting the contribution of each photon hit point onto its surface [Sturzlinger and Bastos 1997]. A splat has the shape of a disc, so the contribution is added to each pixels covered by the projection of the disc into the image and that belong to the same surface. The problem of reconstruction is replaced by a rasterization process. Splatting is performed with graphics hardware using a textured and colored quad. The texture corresponds to a regular sampling of the kernel function and the color is equal to the power of the photons divided by the square of the bandwidth. The final modulated color is then equal to the contribution of the photon. Additive blending is activated to add the contribution of the splat to each pixels. Also, the contribution of a photon must be added only to pixels that belongs to the same surface it hits. The process is shown in Figure 1 at different level of completion.

3.2 Adjustment to low precision

The principles of photon splatting is quite simple but its implementation on graphics hardware involve difficulties. As explained before, the frame buffer is used to store the estimate at each pixels but frame buffer precision is limited to one byte per channel. It is clearly not sufficient. Photon splatting needs a lot of precision: the contribution of a splat is a triplet of RGB encoded in 32 bit floating point by channel whereas the frame buffer of current graphics hardware uses 8 bit fixed point for each channel. The number of non zero contributions from photons hit points are in the order of five hundred so obviously the contribution cannot be scaled to a 8 bit precision. But, using some restriction, frame buffer precision can be sufficient.

Figure 1: Photon Splatting stopped at 0.1% (Left), 1% (Middle) and final (Right).

To adapt to low precision buffer, we choose to use the constant kernel function that is equal to $\frac{1}{\pi}$. Using constant kernel function is not so a drawback contrary to intuition. It has been shown [Silverman 1986] that the kernel function that minimize the mean integrated square error is the Epachenikov kernel K_e. Then, other kernel functions are compared to K_e to measure their efficiency. A remarkable result is that their efficiencies are close to one. The constant kernel has an efficiency of 0.93. So, for an estimation done with the Epachenikov kernel and with n of random points, $n/0.93$ random points are needed to obtain the same level of accuracy.

So, the formula (6) is simplified using the constant kernel function:

$$c(X) = \frac{\phi}{\pi h^2}$$

We will consider for the moment that the power ϕ and the bandwidth h is constant for every splat. In result, the contribution of each photon is constant. So, to reconstruct the irradiance at each pixels, we need to compute only the number of contributions, an integer value that can be stored easily in the frame buffer. Once all the photons have been splatted, the irradiance is reconstructed at each pixels (i, j) using the number of contributions N_{ij} stored in the frame buffer:

$$E(x_{ij}) = N_{ij}\frac{\phi}{\pi h^2} \tag{7}$$

3.3 Choice of the bandwidth

Paragraph 3.2 has shown bandwidth h must be constant to implement photon splatting on a 8 bit frame buffer. In fact, we can choose a constant bandwidth per surface because the splatting is done per surface. The fixed bandwidth h is chosen with an heuristic often used [Shirley et al. 1995] [Sturzlinger and Bastos 1997] :

$$h = C\sqrt{\frac{A}{N}}$$

where A is the area of the surface, N the number of photon hit points on the surface and C a user defined constant, usually between 20 and 30.

So, the following algorithm is used:

1. Build an item buffer to obtain the visible surface identifier per pixel.
2. Compute the bandwidth for each surface.
3. Splats each photons using a quad scaled by the bandwidth of its surface.
4. Read the frame buffer and reconstruct the irradiance for each pixels using (7) with bandwidth equals to corresponding visible surface at pixel.

To improve reconstruction of caustics, we also divide the photons set in two parts : caustic and global photons.The caustic splats use a lower bandwidth, nearly ten times because they are often concentrated on a small part of the scene. So, step 2 to 4 of the preceding algorithm must be repeated for the caustics splats.

3.4 Considerations on photon tracing

As yet, we have considered that the power ϕ is constant. As we compute irradiance in RGB color space, there is one constant power for each channel. To accomplish it with photon tracing, one way is to choose for each photon traced from the light sources a channel R,G or B. The channel c is chosen using the following probability density function $p(c)$:

$$p(c) = \frac{\Phi_T(c)}{\sum_i^{r,g,b}\Phi_T(i)}$$

where Φ is the total RGB power of the light sources. Then, using (2), the power $\phi(c)$ for the channel c is :

$$\phi(c) = \frac{\sum_i^{r,g,b}\Phi_T(i)}{N}$$

This method is inefficient both for photon tracing and image reconstruction because a photon transports power for only one channel. A better solution is that a photon transport a triplet of power for each channel. In this case, the pdf p_e is not anymore function of the wavelength or channel. More, the pdf to choose a light source is replaced by : $f(\Phi_L)/f(\Phi_T)$ where f is a function that transforms the triplet of the power into a real value. f can be either the maximum, the average or the luminance of the triplet. A problem for our method is that ϕ is no longer constant for each splat because some simplifications made in (2) are no longer possible:

$$\phi = \frac{f(\Phi_T)}{f(\Phi_L)}\frac{\Phi_L}{N}$$

The resulting formula is dependent of the selected light source, so the power is no longer constant for each splat.

We must use a different approach for our method. First, light sources with the same self-emitted radiance are grouped together. Instead of sending N photons from the light sources, we sent

Figure 2: Comparison of two images rendered with photon splatting with boundary bias (Left) and without (Right)
Notice the darker regions at boundaries and on small surfaces

$\frac{f(\Phi_G)}{f(\Phi_T)}N$ photons from each group of light sources where Φ_G is total power of group G. Using the pdf $\frac{A_L}{A_G}$ to select a light in the group G, the power ϕ_G for each splat is constant for a given group:

$$\phi_G = \frac{\Phi G}{N_G} \quad \text{with} \quad N_G = \frac{f(\Phi_G)}{f(\Phi_T)}N$$

The rendering of the splats is decomposed for each group of light sources. Furthermore, for each light group, the photons transport the power ϕ_G. If the photon hits a surface, it will be rendered as a splat in the reconstruction pass. Because we are only interested in the number of contributions, the color given to the corresponding splat is $(1,1,1)$. After hitting a surface, the photon is scattered according to the BRDF at the hit point. The probability of scattering or absorption is done for each channel. The resulting power is the product between Φ_G and a mask (c_0,c_1,c_2) where c_i is equals to 1 if scattered, 0 if absorbed. So, if this photon hits a surface again, the color given to the corresponding splat will be equal to the mask.

4 Improving the reconstruction

4.1 Boundary bias

The kernel density estimator assumes that the support of the density function is unbounded. This is not the case in global illumination where the surfaces are bounded. Consequently, energy leaks appear at the boundary of surfaces because irradiance is underestimated as shown in Figure 2. The underestimation of the irradiance is an important visual source of error for small surfaces. In order to overcome this problem, the estimator at pixel (i,j) needs to be scaled by $\frac{A_{ij}}{\pi h_{ij}^2}$ where h_{ij} is the bandwidth at (i,j) and A_{ij} is the intersection area between the support of the estimator and the visible surface at (i,j). The support of the estimator is a sphere of radius h_{ij} centered on the nearest point visible at (i,j). The computation of the intersection area is a quite complex geometric problem for any kind of input geometry. An implementation with triangle meshes is presented. The algorithm is based on connected triangle mesh:

1. Find a triangle inside the sphere
2. Find the circle deduced from the intersection of the plane of the triangle with the sphere
3. Add the intersection area between the circle and the triangle to the final result
4. Repeat since step 2 on adjacent triangles if their shared edges cross the circle

To avoid cyclic graph on the connected triangle mesh, a mailbox is used on the triangles to know if they have already been visited. It works on complex surfaces as shown in Figure 5.

So, we need to compute the intersection area between a circle and a triangle. First, the triangle is clipped against the circle. In result, a polygon is obtained if the triangle intersects the circle (filled with light gray in Figure 3). The clipping of the triangle introduces segments that are not parallel to the original segments of the triangle. These new segments defined half planes. The intersection area between these half planes (filled with dark gray in Figure 3) and the circle is computed and added with the area of the polygon to obtain the intersection area between the circle and the triangle.

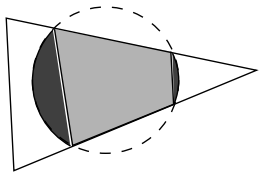

Figure 3: Intersection area of a triangle with a circle

4.2 Adaptive image reconstruction

Using a fixed bandwidth per surface blur discontinuities of the irradiance as for caustics and shadows for example. Different approaches exist for this problem [Myszkowski 1997] [Walter 1998]. But with our method we cannot use a different bandwidth per photon. But we can generate different images with different bandwidth and choose the best estimator between the set of images. The following algorithm is used to allow different bandwidths per pixels:

1. Generate an image estimate using fixed bandwidth per surface

2. For each pixels, an oracle is used to know if the current estimate is added to the final image otherwise the pixel is tagged as incorrect

3. If any incorrect pixels remains, compute a new image estimate with the fixed bandwidth reduced by a certain amount and repeat step 2

The fixed bandwidth per surface is reduced by one quarter for each subsequent image. The result of the adaptive reconstruction is presented in Figure 4. The adaptive reconstruction has only been done on the caustics splats for efficiency reason. The number of caustics splats is generally sufficiently low compared to the total of splats so the cost of rendering many images is not prohibitive.

4.3 Oracle for bandwidth selection

The oracle should decide for each pixels if its current estimator gives a reasonable level of noise and bias. An often used heuristic to estimate the level of noise is based on the number of non zero contributions from the data points. Fortunately, we know this number for each pixel after the splatting phase. So noise can be estimated easily.

To measure bias, our method is based on the fact that the bias of a density function $f(x)$ is proportional to $h^2 f''(x)$ [Silverman 1986]. From this property, we have deduced that a good way to control the bias would be to limit the bandwidth only in pixels where the second derivative of the estimator is important. As seen in 3.2, the only parameter that vary along the image is the number of non zero contributions N_{ij} . So, it is sufficient to compute the discrete second derivative of the N_{ij} using finite differences. Finite differences are used to compute the partial second derivative in x and y in the image space. Then the maximum value of the partial derivative is taken to measure the bias level. The points p_{ij} of the pixels (i, j) are not regularly spaced in world space. So, we use the formula of second derivative on an irregular grid shown here for the x axis:

$$\frac{\partial^2 N}{\partial x^2} = \left(\frac{N_{i+1j} - N_{ij}}{h_{ij}} - \frac{f_{ij} - f_{i-1j}}{h_{i-1j}} \right) \frac{h_{ij} + h_{i-1j}}{2}$$

where h_{ij} is the distance between the point p_{ij} and p_{i+1j}. This is the central difference formula, but on the boundaries, the backward or forward difference is used.

Our oracle must decide if decreasing the bandwidth will give lower error. First, if variance is already too high, answer is negative. Then, if variance is low, decreasing the bandwidth will decrease bias so answer is positive. Finally, if variance is in the middle range, the value of the second derivative is computed to decide if it is a region of high bias.

Finally, three parameters control the oracle : N_{min} and N_{max} (the minimum and maximum number of non zero contributions) and ε (the threshold for the bias). So, the oracle uses the following conditions for a pixel (i, j) : if N_{ij} is less than N_{min} or N_{ij} is less than N_{max} and the level of bias is under ε then the pixel is correct.

This oracle is quite efficient, the only computation is finite differences formula which is not complex.

5 Implementation

5.1 Frame buffer overflow

The number of contributions is about five hundreds to two thousands, so it overflow the 8 bit value of each channel. To resolve this problem, we need to add periodically the content of the frame buffer to a higher precision buffer after a number of rendered splat and to reset the frame buffer. In OpenGL, the accumulation buffer allows effectively to add the current value of the frame buffer to a higher precision buffer. Unfortunately, it is not supported on many graphics cards and it is very slow if done in software by the drivers. In fact, reading the frame buffer and storing the result in a higher precision buffer at application level is faster than using the software alternative of accumulation buffer provided by the drivers.

Figure 5: An example of photon splatting with a complex surface and boundary bias correction

To achieve good performance, the data transfer between the graphics card and the CPU memory must be minimized. It is difficult to guess an a priori upper bound on the maximum number of splats that can be rendered before causing a buffer overflow. So, an heuristic is used to decide when to read the frame buffer. The idea is to compute an upper bound on the maximum number of non zero contributions in the frame buffer for the current scene. Then, we divide this number by the maximum value of the frame buffer to obtain the number of steps in the rendering process.

We approximate the probability that a splat adds its contribution to a pixel as the area of the splat over the total area of the scene. So, the maximum number of contributions $\tilde{N}$ per pixel is approximated as:

$$\tilde{N} = n \frac{\pi h^2}{D A_t} \tag{8}$$

where n is the total number of photons hit points, A_t is the total surface area of the scene, h the average bandwidth of the surfaces and D a constant between 0.0 and 1.0. Then, we decompose the photon splatting in $\tilde{N}/M$ steps, where M the maximum value in the frame buffer

The constant D is here to take into account that a splat has not equal chance to fall into each region of the scene. Some regions can be occluded with regard to light sources for example. So, D must be approximatively proportional to the percentage of occluded area in the scene. This approximation works if the photon hit points are well distributed in the unoccluded scene. This assumption is violated for caustics lighting effects where the density of photons hit points is high in a particular region. The caustics splats are rendered with a very low value of D as 0.1 for example which means caustics appears only in 10 percent of the scene.

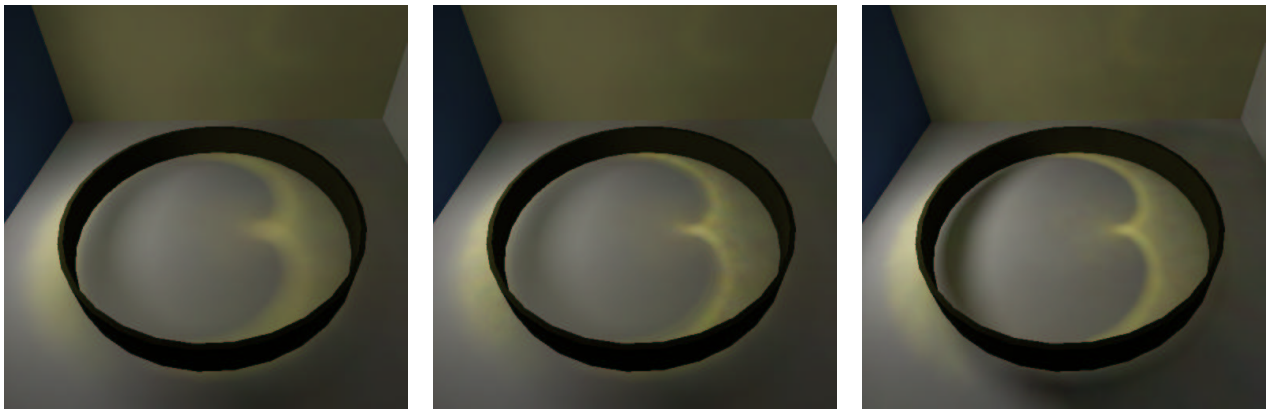

Figure 4: Three images rendered with Left : fixed bandwidth, Middle : adaptive bandwidth, Right : reference solution

5.2 Hardware implementation

The rendering of the splat must be restricted to the visible part of its surface. An implementation with the OpenGL API is presented here.

When OpenGL rasterize a geometric primitive, it converts it into fragments. A fragment corresponds to a pixel in the frame buffer. A series of operations are performed in the fragment before being stored in the frame buffer. The interesting operations for our purpose is the several tests a fragment has to pass before going in the frame buffer. Depth test can be used to limit the rendering of the splat to the pixels with same depth values but the precision is not sufficient to test equality even between a splat and a planar surface. Stencil test compares a reference eight bits value to the stencil pixel value. Using equality comparison, stencil test can effectively restrict the rendering of the splat to surfaces with the same identifier but we are limited to only 255 different surfaces.

Fortunately, the new generation of graphics hardware provides programmable fragment operations often called pixel shader or fragment shads. Several instructions are provided to the programmer to output the final color that will be stored in the frame buffer. Instructions operates on different RGBA colors registers $r0, r1, ..., rn$ with input as the primary color or texels fetched from texture units. Only one conditional instruction exists : a greater or equal test to a value of 0 or 0.5. So, our idea is to use the fragment shader to compare the surface identifier of the splats (SIs) and the surface identifier of the current pixel (SIp). More precisely, fragment shader will return a black color if the comparison is false else it will return the product of the kernel texture and the color mask.

To perform the comparison, the fragment shader needs as input SIs and SIp. SIs is passed to the shader as the secondary color. For SIp, a texture is created with the same size than the image, each texel value corresponds to the surface identifier of the pixel. We called this texture the surface identifier texture (SIT). To obtain SIp, SIT must be looked up using the same coordinate (i, j) than the pixel on which the fragment shader operates. To accomplish that, the texture coordinate of SIT are generated using the automatic texture coordinate generation provided by OpenGL. If texture coordinate generation is activated, the texture coordinate of each vertices is equal to its position coordinate transformed by a matrix given by the programmer. So, we pass the transformation matrix from world to screen used by OpenGL for computing the image to the texture coordinate generation stage. In result, for each fragment, the texel fetched from this texture unit is equal to SIp.

So, the following step are needed to prepare the OpenGL pipeline to the splatting pass:

- Build an item buffer : each triangle is rendered with a color equals to its surface identifier
- Read the frame buffer to build the surface identifier texture
- Build the kernel texture
- Activate the kernel texture on unit 0
- Activate the surface identifier texture on unit 1, with automatic texture coordinate generation activated as explained
- Activate the fragment shader

Figure 6: *Cabin* scene with 1 million of photons.

The following sequence of instructions must be performed by the fragment shader to implement the comparison:

$$r0 = tex1 - col1 + 0.5$$
$$r1 = (r0 >= 0.5)?1 : 0$$
$$r0 = col1 - tex1 + 0.5$$
$$r2 = (r0 >= 0.5)?1 : 0$$
$$out = r1 * r2 * col0 * tex0$$

where col1 is the secondary color (SIs), tex1 the texel of the second unit (SIp), col0 the primary color (the color mask), and tex0 the texel of the first unit (kernel texture).

In fact, the method presented works only with the `ATI_fragement_shader` extensions only available on ATI Radeon card. We have implement hardware accelerated photon splatting on a NVIDIA GeForce3 using `NV_registers_combiners` extensions. With this architecture, the comparison can be performed only on the alpha portion of one register. Because of this restriction, the comparison in the fragment shader has only been done on the blue and alpha channel, so it allows only 16 bit identifier but it is sufficient on all scenes tested. We have preferred to present the possibility with `ATI_fragement_shader` extensions because it is simpler and closer to the future API of graphic card as OpenGL2.

6 Results

In this section, we demonstrate our approach with several scenes from the MGF file format. Three scenes with different complexity are used:

- *Caustics* : a simple scene with caustics cast by a reflecting ring
- *Conference* : a conference room with high geometric complexity (90 000 triangles)
- *Cabin* : a scene with many rooms (45 000 triangles)

Scene	PT	NS	PS1	PS2	IR
Caustics	0.91s	178901	1.06s	2.48s	0.4s
Conference	9.13s	160516	1.23s	3.06s	1.8s
Cabin	4.76s	187184	1.17s	2.57s	1.2s

Table 1: Comparison with three scenes, PT : photon tracing, NS : number of splats, PS1 : photon splatting for a 512x512 image and PS2 : photon splatting for a 900x900 image , IR : image reconstruction

All results were obtained with a 1.2GHz Athlon processor, 512 Mb of memory and a GeForce3 graphic card. First, results in Table 1 are presented for a coarse solution (100 000 photons traced from the light sources) using fixed bandwidth reconstruction. The results of the photon splatting pass seems slow compared to the specification of the graphic card. In opposition to first belief, the bottleneck is not only the frequent reading of the frame buffer. It is only responsible of one third of the rendering time even if there is ten to twelve reading of the frame buffer. We think the main bottleneck is the huge overdraw of the splats. Clearly, our method is fillrate limited.

Method	1e5	1e6	1e7
Fixed	1.23s	5.68s	15.06s
Adaptative	7.3s	14s3	39s

Table 2: Simulation on the *Conference* scene with different number of photons and the two methods

As expected, the scaling with scene complexity of the photon tracing pass is logarithmic. The increasing time with respect to the geometric complexity during the image reconstruction pass is due to the computation the intersection area between the kernel support and the surface. Otherwise, the splatting of the photons is totally scene complexity independent. It shows clearly that our method is scalable with scene complexity.

In Table 2, some results are shown with the Conference scene but with different accuracy for the light transport pass. The time of adaptive reconstruction are given for a maximum of four decreasing bandwidth. These results shows that the complexity of the reconstruction pass does not depend only on the number of splats and the size of image but also depends on the average size of the splats in the image. It is shown in Table 2 because between a simulation with 100 000 photons and one million, there is tenth more splats but the increase in time of the reconstruction is much lower. Also, the simulation with 10 millions of photons uses disk caching. It explains the relative increase in time. Disk caching is done directly by our software. Because the photons are accessed linearly, it is very easy to implement. In that case, the photons are stored in only 64MB to test the efficiency of disk caching.

7 Conclusion

In this paper, we have presented a two pass method to compute an irradiance image of a scene. The image can be recomputed efficiently for different viewpoints. Our method has several advantages. First it is very scalable with respect to the geometric complexity. The photon tracing pass has logarithmic complexity and the reconstruction does not depend strongly on the geometric complexity. Another advantages is that our algorithm uses computer resources often neglected by other global illumination algorithms : graphics card and hard drive storage. Our method will certainly takes advantage of the increasing computational power of graphics hardware. Moreover, next generation of consumer card will support more precision for the frame buffer which is positive for the accuracy of our algorithm. On the other hand, our method is fillrate limited, so using a higher precision buffer will also decrease efficiency.

Also, we plan to integrate photon splatting in a more general global illumination algorithm. Indeed, to integrate effect as specular reflections and refractions, photon splatting can be used with a ray tracer. Particuraly, image based ray tracer [Lischinski and Rappoport 1998] seems a promising approach to compute the full light transport paths.

Figure 7: *Conference* scene with 1 million of photons and adaptive reconstruction.

References

CHEN, S. E., RUSHMEIER, H. E., MILLER, G., AND TURNER, D. 1991. A Progressive Multi-Pass Method for Global Illumi-

nation. In *Computer Graphics (ACM SIGGRAPH '91 Proceedings)*, vol. 25, 164–174.

CHRISTENSEN, P. H. 1999. Faster photon map global illumination. *Journal of Graphics Tools 4*, 3, 1–10.

GRANIER, X., DRETTAKIS, G., AND WALTER, B. 2000. Fast global illumination including specular effects. In *Rendering Techniques 2000 (Proceedings of the Eleventh Eurographics Workshop on Rendering)*, Springer Wien, New York, NY, B. Peroche and H. Rushmeier, Eds., 47–58.

HECKBERT, P. 1990. Adaptive Radiosity Textures for Bidirectional Ray Tracing. In *Computer Graphics (ACM SIGGRAPH '90 Proceedings)*, vol. 24, 145–154.

JENSEN, H. W. 1996. Global Illumination Using Photon Maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, Springer-Verlag/Wien, New York, NY, 21–30.

KAJIYA, J. T. 1986. The Rendering Equation. In *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, vol. 20, 143–150.

LISCHINSKI, D., AND RAPPOPORT, A. 1998. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques '98 (Proceedings of the Ninth Eurographics Workshop on Rendering)*, 301–314.

MYSZKOWSKI, K. 1997. Lighting reconstruction using fast and adaptive density estimation techniques. In *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, Springer Wien, New York, NY, J. Dorsey and P. Slusallek, Eds., 251–262. ISBN 3-211-83001-4.

SHIRLEY, P., WADE, B., HUBBARD, P. M., ZARESKI, D., WALTER, B., AND GREENBERG, D. P. 1995. Global Illumination via Density Estimation. In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, Springer-Verlag, New York, NY, P. M. Hanrahan and W. Purgathofer, Eds., 219–230.

SILVERMAN, B. 1986. *Density estimation for statistics and data analysis*. Chapman and Hall, New York NY.

STURZLINGER, W., AND BASTOS, R. 1997. Interactive rendering of globally illuminated glossy scenes. In *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, Springer Wien, New York, NY, J. Dorsey and P. Slusallek, Eds., 93–102. ISBN 3-211-83001-4.

SUYKENS, F., AND WILLEMS, Y. D. 2000. Density control for photon maps. In *Rendering Techniques 2000 (Proceedings of the Eleventh Eurographics Workshop on Rendering)*, Springer Wien, New York, NY, B. Peroche and H. Rushmeier, Eds., 23–34.

VEACH, E., AND GUIBAS, L. J. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Computer Graphics Proceedings, Annual Conference Series, 1995 (ACM SIGGRAPH '95 Proceedings)*, 419–428.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Computer Graphics (ACM SIGGRAPH '97 Proceedings)*, vol. 31, 65–76.

WALTER, B. J. 1998. *Density Estimation Techniques for Global Illumination*. PhD thesis, Program of Computer Graphics, Cornell University, Ithaca, NY.

WARD, G. J., AND HECKBERT, P. 1992. Irradiance Gradients. In *Third Eurographics Workshop on Rendering*, 85–98.