Sensor based Lighting

Mariano M. Banquiero, Matias N. Leone,
Explotación de GPUs y Gráficos Por Computadora - GIGC
Departamento de Ingeniería en Sistemas de Información, Universidad Tecnológica Nacional
Buenos Aires, Argentina
Email: {mbanquiero, mleone}@frba.utn.edu.ar

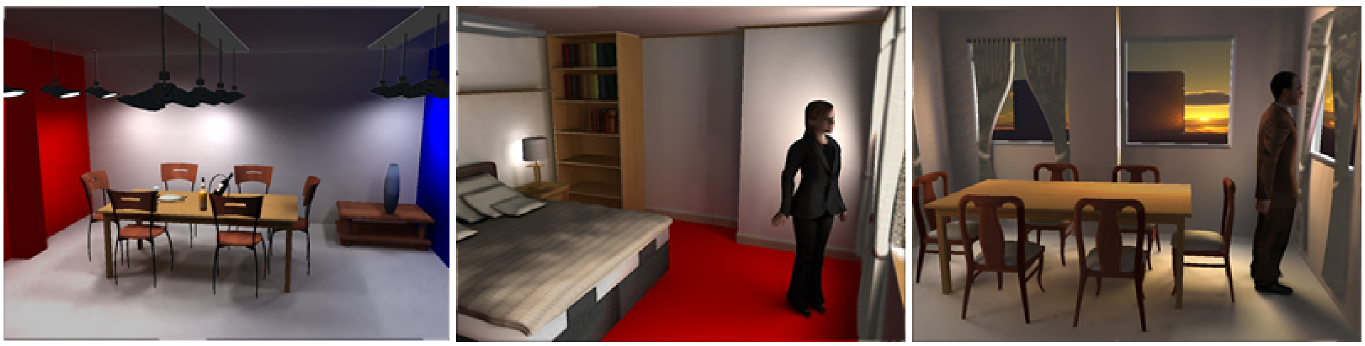

Fig. 1. Three indoor-scene renders computed by the Sensor based lighting technique

***Abstract*—Commonly used global illumination techniques have good quality but are computationally expensive and difficult to parallelize in GPU. We propose a novel global illumination algorithm that locates virtual cameras called sensors in a carefully selected set of visible pixels of the scene. Each sensor captures the incoming light from every direction of the scene and produces an estimation value that is later compose with the original image. The algorithm is an offline solution implemented in GPU in a very parallelizable way, supports arbitrary-form light sources, and produces soft shadows, ambient occlusion and color bleeding effects.**

***Keywords*-Radiance estimation; GPU; Global illumination;**

I. Introduction

Many techniques used in computer graphics to simulate lighting effects are based on source lights called point-lights. They are fast to compute and easy to integrate in a real-time graphics pipeline.

But real life lighting appliances are normally design to avoid direct lighting because its uncomfortable for the human eye. For example, a table lamp came usually covered by a surface that produces a glimmering light in the room. Sun light entering a building through a window with curtains behaves like a large superficial light source.

These arbitrary-form light sources behave in a very different way of commonly used point lights, for example when generating soft shadows with complex penumbra area.

Furthermore, in a closed room, the light coming from different light sources will bounce against many surfaces before finally reaching the eye. En each of this bounces part of the energy is absorbed and another part is reflected in certain wave length, according to the BRDF [1] distribution. The human eye is particularly sensitive to this kind of indirect illumination. This way, a red wall may transfer some of its color to a white ceiling, producing a reddish look, known a color bleeding.

Contributions: The aim of this paper is to present an offline algorithm implemented in GPU that supports all this subtle effects, from a physically based perspective: an algorithm capable of modeling a large amount of arbitrary-form source lights, and capable of producing physically correct soft shadows, besides able to approximate another aspects of global illumination, like color bleeding and ambient occlusion.

II. Related work

Many algorithms have been developed to generate realistic images based on global illumination models. Path tracing [2], Monte Carlo ray-tracing [3] and Metropolis Light Transport [4] are examples that used random sampling to estimate a solution that converge to a physical correct result. But they tend to required thousands of samples per pixel in order to achieved good results, which makes them computationally expensive.

Radiosity [5], a finite based element solution, also converges to a physically correct solution with an iterative procedure. It is also a computationally expensive and it requires the scene to be fairly tessellated.

All techniques mentioned below has the objective to produce a physically correct image regardless of the time required to be computed. On the other side, there are algorithms acutely fast to be used in real-time applications. They can compute direct shadows (shadow map [6], volume shadows [7]) and ambient light (ambient occlusion [8]) in a very simplified form. These techniques are usually known as to be perceptually correct but physically incorrect. This means that although they are not accurate in physics terms they can be visually attractive to the human eye.

Many methods exist to also produce soft shadows perceptually correct, like Variance Shadow Maps [6] and Cascaded Shadow Maps [7]. Recently Radiosity and others global ilumination algorithms have been adapted to execute in GPU, some of them even at real-time frame rates. The technique Instant Radiosity [9] constructs random paths from the light sources and creates new virtual point lights *(VPLs)* where these paths encounter a surface. The work described in [10] uses an hemispherical proyection to capture radiance in an path texture.

III. Sensor Based Lighting

A. Algorithm overview

The technique presented in this work consists in positioning virtual cameras in many visible points of the scene, Fig. 2. These cameras act like sensors that measure the amount of incoming light from all directions. In order to achieve that, each camera draws the complete scene in an auxiliary texture. The texture's size is big enough to get every illumination details, and then is reduced to unique average value which represents the light received from the entire scene. This value is stored in an output texture used later to draw the final scene.

B. Radiometry equations

The rendering equation [2] describes the transport of light from one surface point to another as the sum of emitted radiance and reflected radiance. Because light transport is usually perceived in equilibrium, the equation describes the steady-state radiance function for the given scene [2].

$$\underbrace{Lo(p,w)}_{\text{Outgoing radiance}} = \underbrace{Le(p,w)}_{\text{Emitted Radiance}} + \underbrace{Ls(p,w)}_{\text{Scattered Radiance}}$$

The term Le forms part of the scene description and correspond to the light sources, which are the only objects that emit light. The second term, the scattering equation, is of more significant for this work:

$$L_s(p,w_o) = \int_\Omega L_i(p,w_i) f_s(p,w_i \rightarrow w_o) d\omega^\perp$$

Where wo is the view direction, wi the incoming direction, $d\omega^\perp$ is the differential projected solid angle and the integral is made all over the positive hemisphere above the surface. The fs term is the BRDF [1] and represents the amount of radiance coming from the wi direction that reflects towards the wo direction. In the simplest case the function can be constant (perfect diffuse surfaces) and the fs becomes a $\frac{c}{\pi}$ factor out of the integral [11]. c is constant ≤ 1. This reflects the fact that the radiance reflected out of the surface can not be greater than the inconming radiance.

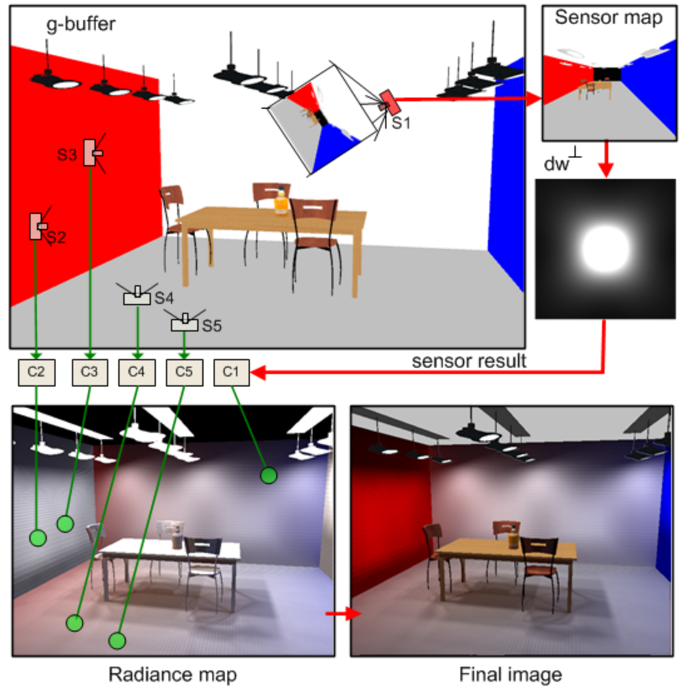

Fig. 2. Census function. The light coming from the scene is captured by the top right sensor in the first image. The sensor's image is then weighted by its incident angle, and the last step is to average the final color.

The ingoing radiance can be expressed in terms of outgoing radiance using a ray casting function [12], and the simplified scattering equation is:

$$L_s(p,w_o) = \frac{c}{\pi}\int_\Omega L_o(ray(p,w),-w)d\omega^\perp$$

The scattering equation can be used to predict the surface color from the incident light [13]. Our work applies this principle to adapt the equation to a GPU suitable computation, Fig. 3.

C. Outgoing radiance estimation

Lo appears in both sides of the rendering equation, forming a recursive expression. The strategy of this work is to replace Lo by an estimator to avoid the recursive form. Considering a fixed direction wi and the point $q = ray(p, wi)$ then two situations must be studied:

A If q is over the surface of a light source Le (emitted radiance) it will have a value corresponding to the color emitted by that light, an information that forms part of the scene. We can assume that this value is highly to Ls for a light source. So then we can express:

$$Lo = Le \tag{1}$$

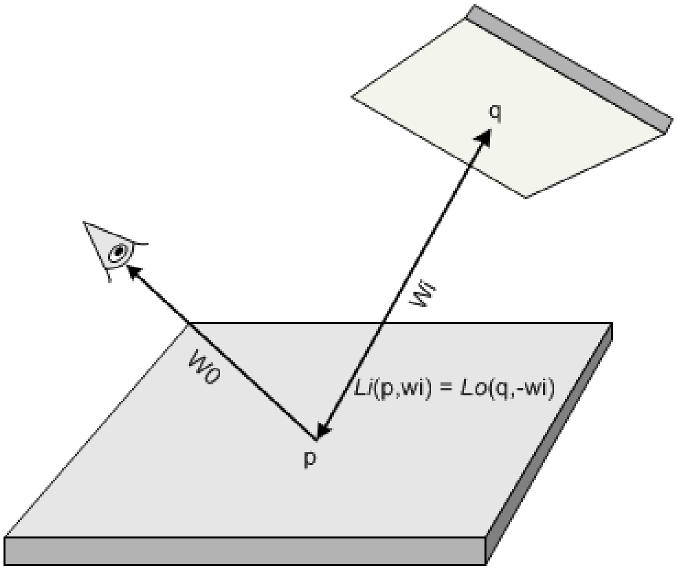

Fig. 3. Ligth Transport. Ingoing Radiance (Li) can be expressed in term of Outgoing Radiance (Lo). The radiance coming to p from a direction wi is the radiance exiting from q trhough the direction $-wi$. The point q is the first intersection between the ray from q towards wi direction and the rest of scene.

B If q is over a surface that reflects light, Lo is described by the scattering equation, then its value depends on the light coming from all directions of the hemisphere. In this case the surface diffuse color (also part of the scene description) is taken as an estimation of the value, multiplied by an epsilon e:

$$Lo = D.e \tag{2}$$

The epsilon term is a small number that allow setting the importance of the ambient light related to the direct light. And it represents the fact that the incident light is usually reflected in many directions, so the amount of light reflected in one particular direction is very small [1].
So the Lo estimator is expresses as:

$$L_o^* = Color(q)K(q)$$

The estimator groups together 1 and 2 in the same equation. From a computing point of view the difference between 1 and 2 is just the specific value of $K(p)$. In one case is the light associated energy and in the other is a small epsilon. But from physical point of view the two cases correspond to totally different processes. It is important to note that the calculated values may take a wide range of results. So the precision of a high dynamic range [14] is required. The term $Color(q)K(q)$ will be shortened as $Ck(q)$.

D. Scattered radiance estimation

Replacing the Lo estimator in the scattering equation and approximating the integral with a summation we get 3:

$$L_s(p, w_0) = \frac{c}{\pi}\int_{\Omega} Ck(q)d\omega^{\perp} \sim \frac{c}{\pi}\sum_{t=1}^{n} Ck(q_t)d\omega^{\perp} \tag{3}$$

E. Sensor

In this algorithm a sensor represents a virtual camera that allows us to capture the incoming light from all directions of the scene. This light is then stored in texture named *sensor map*, that represent the sensor area. The camera is located at the required point p in the direction normal to the surface of the scene, Fig. 4.

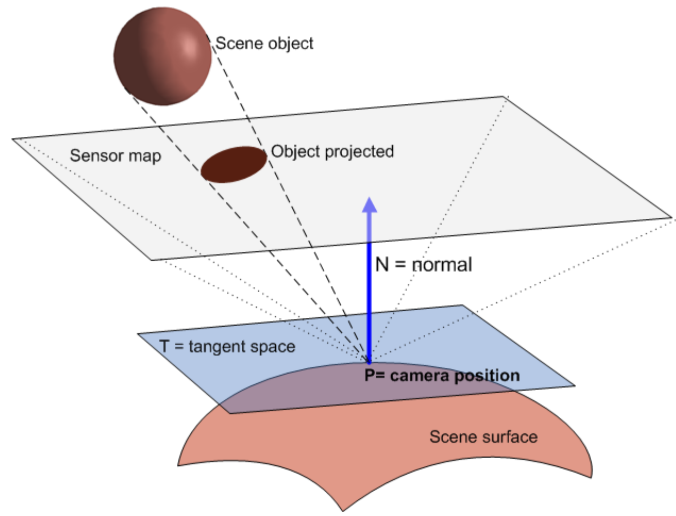

Fig. 4. Sensor setup. The virtual camera is located at the required point p in the direction normal to the scene surface.

The *sensor map* size is $rd_x \times rd_y$ and each texel represent a sample of the Ck term for given direction in space. So the sensor is applied to compute the Ck term for a large quantity of samples and then to use them in the equation 3.
In order to be able to capture the scene illumination from most possible angles, the camera field of view (fov) is of 135 degrees (which amplitude is higher than the 45 degrees commonly used). This brings out two main problems. Firstly, such a high value in fov amplitude tends to loose precision at the center of the image, just where the samples have more incidence. Secondly, we can never take a sample of 180 degrees of amplitude using a perspective or orthogonal projection. This means the sensor is incapable of capturing the light coming from angles out of the fov.
The Ck term for a giving pixel, drawn from the sensor point of view, can be computed using a Pixel Shader in GPU because both $Color$ and K are part of the scene description.
The original summation over the amount of samples is replaced by a double summation to iterate over the values of the *sensor map* in width and height.

The remaining part of the equation, $dw^{\perp}$ (differential projected solid angle), can be obtained by simply projecting the area of the unit sphere cover by the dw (differential solid angle) orthogonally onto the tangent space, and then finding the area of the resulting planar region.

As we can see in image Fig. 5:

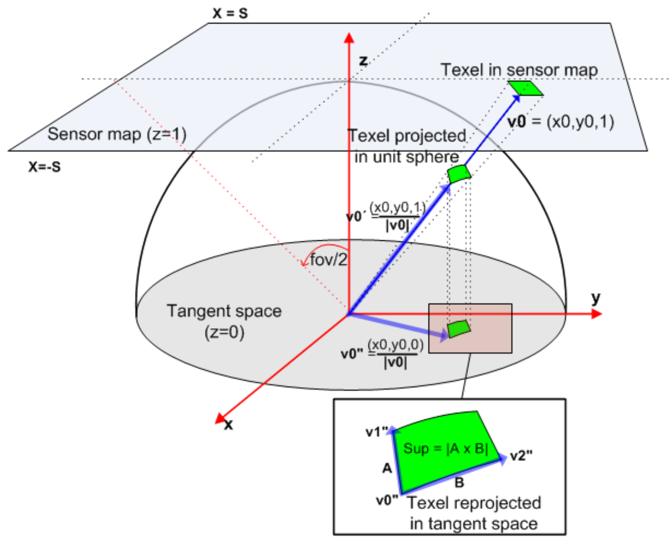

Fig. 5. Solid angle measures and projected pixel area. A texel area in the *sensor map* is first projected in the unit sphere and then projected again in tangent space. This projected area is used to measures the contribution of the sample.

$$S = tan(\frac{fov}{2})$$

$$x = 2S(\frac{j}{rd_x} - 0.5)$$

$$y = 2S(\frac{i}{rd_y} - 0.5)$$

An arbitrary point $v = (x, y, 1)$ in the *sensor map* is first normalized to proyect it in the unit sphere, and then is proyected again in tangent space by simply setting $z = 0$.
We denote this with the v" operator:

$$V'' = \frac{(x, y, 0)}{\|(x, y, 1)\|}$$

Given a sample (i,j) in the sensor area, we call v0,v1 and v2 to the vertex:

$$V_0 = (x_j, y_i, 1), V_1 = (x_{j+1}, y_i, 1), V_2 = (x_j, y_{i+1}, 1)$$

The projected pixel area in tangent space can be approximated using de cross product:

$$d = ||(V_1'' - V_0'') \times (V_2'' - V_0'')|| \tag{4}$$

The value 4 represents the pixel projection over the tangent space in arbitrary pixel units. The sum of the area of all possible angles is equal to the whole disc surface hemisphere, which equals π. So the relative value of each sample (without units) is:

$$\frac{\|(V_1'' - V_0'') \times (V_2'' - V_0'')\|}{\pi}$$

The total area of the samples can be calculated as $rd_x.rd_y$, which allows us to approximate the term (4) with:

$$F_{ij} = \frac{\|(V_1'' - V_0'') \times (V_2'' - V_0'')\|}{\pi}.(rd_x.rd_y)$$

With this formulation, the energy values associated with each source light must be set in [radiance units] $\times$ pixel.

As can be seen the F_{ij} values only depend on i and j, so they can be pre-computed. The physical interpretation means that the light coming from similar directions to the normal has the max incidence over the sensed value, while the peripheral directions decreased rapidly.

The estimated scattering equation becomes:

$$L_s(p, w_o) \sim \sum_{j=1}^{rd_x} \sum_{i=1}^{rd_y} c_{ij} F_{ij}$$

This value is the one obtained as a result in each sensor and then stored in the output texture called *radiance map*.

F. Visible points determination

The first step consists of creating a geometry buffer to store some useful information of the visible points of the scene: position, normal and polygon or mesh id.
Different packing strategies derived from the deferred rendering techniques [15] can be used to create this *G-buffer*.

G. Sensor quantity and distribution

In many parts of the domain the value of Ls changes in a soft way. Except when there is a change of surfaces, it is very likely that Ls will be continuous in the point neighborhood. This suggests that it is not necessary to compute Ls in all the image points. The algorithm can being sensing at regular intervals from a given tile of $dt \times dt$. Each tile will have a sample to which Ls is estimated. Then for each sample it computes the variation with the four closer samples, Fig. 6. If this variation is less than a given threshold, called $dradiance$, the whole set can be estimated with interpolation between the four neighbors. By the contrary if the variation is greater than $dradiance$ then more samples may be located in tiles of half the size of the initial interval $(\frac{dt}{2})$. The process is repeated until only final pixels are left, where Ls is computed specifically to complete the scene, Fig. 7. In this way the algorithm converge to the solution in a progressive way, using more sensors where the image needs them most.

H. Final image composition

So far we have generated a map of values for Ls for each visible point of the scene, which are stored in the *radiance map*. Each value is called $R(x, y)$. Next, the texture is combined with the original image in order to produce the final image. For a given point (x, y) of the original image,

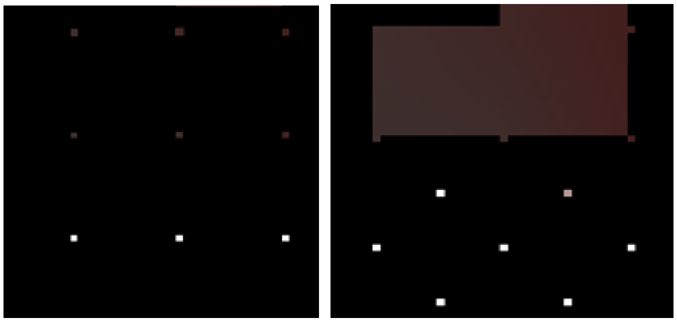

Fig. 6. The red samples have little variation between them and can be interpolated to drastically reduce the amount of sensing operations. The white samples however have an abrupt change and require more samples.

the same point of the *radiance map* contains the value of Ls. Both values are multiplied:

$$final(x,y) = color(x,y)R(x,y)$$

The tile optimization of the algorithm tends to produce small discrete blocks in the final image, an aliasing normally seen in most sampling techniques that use regular intervals. Replacing the regular sampling by a random one (for example with jittering techniques) will transform those aliasing in noise, and the visible artifacts will be appreciable. A better solution is to apply a Gaussian filter [16] to image composition. Fig. 8. A certain radius of samples of the *radiance map* is taken, provided that they belong to the same surface. This restriction is vital in order to avoid summing values of Ls coming from different regions of space, which will produce hard artifacts in the solution. Thats why the *G-Buffer* stored the polygon id.

$$final(x,y) = color(x,y)[\sum_{i=-r}^{r}\sum_{j=-r}^{r} k_{ij} Ls(x+i, y+j)]$$

kij is the relative weight of each sample and is computed considering the total amount of samples that satisfied the surface id condition (that is they belong to the same polygon as the point (x,y)).

$$k_{ij} = \begin{Bmatrix} \gamma_{ij} & \text{si} & id(x+i, y+i) = id(x,y) \\ 0 & \text{si} & id(x+i, y+i) \neq id(x,y) \end{Bmatrix}$$

$$\sum_{i,j=-r}^{r} \gamma_{ij} = 1$$

I. Shadow computing

The size of the *sensor map* $(rd_x \times rd_y)$ must be large enough to capture all the details from the light sources. An area too small may generate discontinuities in the estimation of Ls which will be perceived as noise in the image.

For superficial light sources, or for large enough arbitrary forms, this estimation can be extremely accurate when the proper texture size is given. This is due to the fact that the projection of the light source over the sensor area occupies a large amount of samples. So the sensor is very sensitive to obstacles that may block part of it, producing physically correct shadow penumbras, Fig. 9.

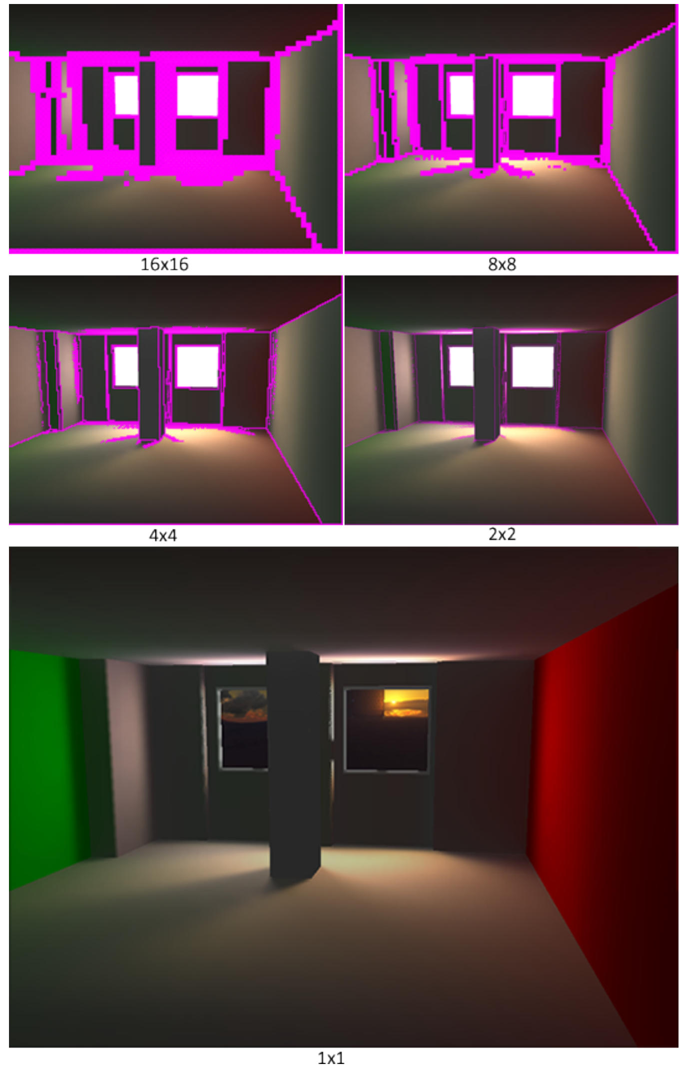

Fig. 7. Tiles progressive computation. The first image shows tiles of 16×16 pixels. Only one sample per tiles is computed which value is interpolated between the neighborhood samples. In the next steps the tile size is reduced. At the end only the edge pixels remain to be sensed, where Ls is discontinuous.

IV. Implementation and Results

The algorithm was implemented using C++ with DirectX 9 and Shader Model 3. Three different metrics were taken in order to understand the behavior of the technique. The total time spent by the algorithm depends on many factors (screen resolution, *sensor map* size, geometry complexity of the scene) So a better indicator for the metrics is what we called the sensor time, which is defined as the total time divided by the amount of sensors required for that scene.

The first metric, Table I, measures the sensor time as the screen resolution is increased. The *sensor map* size is

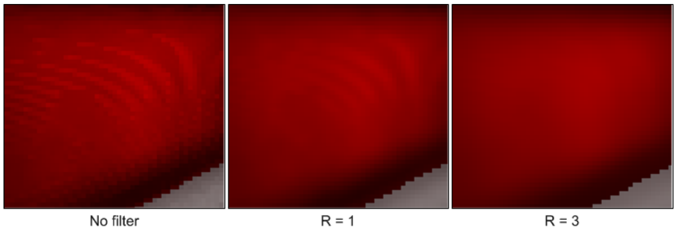

Fig. 8. Gaussian filter application in image composition. Note that the red surface edge (wall) and the gray one (floor) continue to be sharp. The filter is specifically design to preserve the id sudden changes.

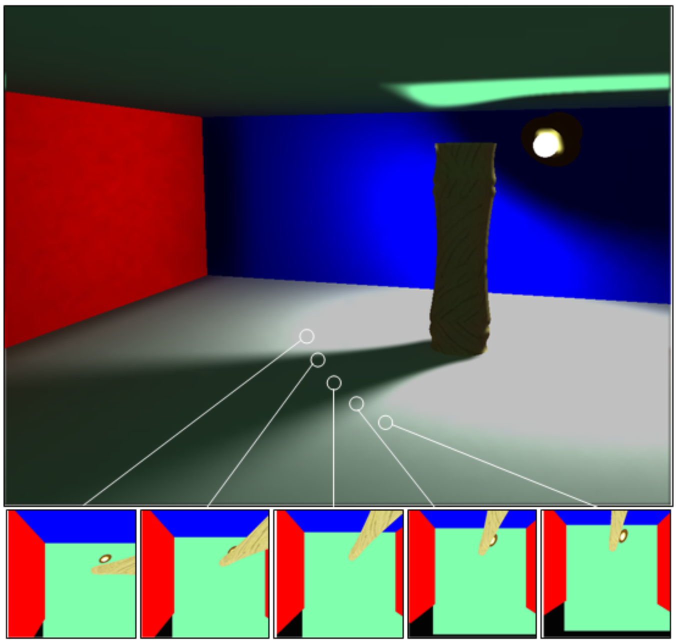

Fig. 9. Shadow computing. The red rectangle represents sensed area. The first screenshot shows the light source totally visible. Then, the visible area of the same gradually decreases as the shadow becomes more evident in the final image. In the fourth screenshot the light source is no longer visible and the area is completely in shadows.

fixed for this test. The conclusion is that time spent for each sensor is constant for a fixed *sensor map* size, and it does not depends on the screen resolution.

The second metric, Fig. 10, measures the sensor time as the *sensor map* size is increased. Three different screen resolutions are taken as reference. The conclusion is that the *sensor map* size influeces directly over the total processing time of the algorithm, almost in a quadratic fashion. And it does not depends on the screen resolution.

The third metric, Fig. 11, measures the amount of sensors required by the scene as the *sensor map* size is increased. The same three screen resolutions as metric two are taken. The amount of sensors is high when the sensor map is small, because each sensor has not enough resolution to capture the scene details, so more sensors are required. But after increasing the size to 128×128, the sensors count begins to stabilize. In our tests, we achieved a good comprimise beetween quality and performance with a *sensor map* of 256×256, Fig. 12. The tests performed also show us that the total time required by the algorithm is much more sensible to the amount of sensors than to the geometry complexity of the scene.

TABLE I
FIRST METRIC: SENSOR TIME RELATED TO SCREEN RESOLUTION.

Screen resolution	Sensors count	Time/sensor (ms)
200x200	15240	4,455
200x200	15240	4,455
300x200	23762	4,439
400x300	40271	4,499
500x300	44300	4,496
500x400	55904	4,541
600x400	63074	4,581
600x500	68276	4,531
700x500	79158	4,587
700x600	83423	4,588
800x600	96266	4,588
900x600	108239	4,578
900x700	112436	4,581
1024x768	133822	4,569

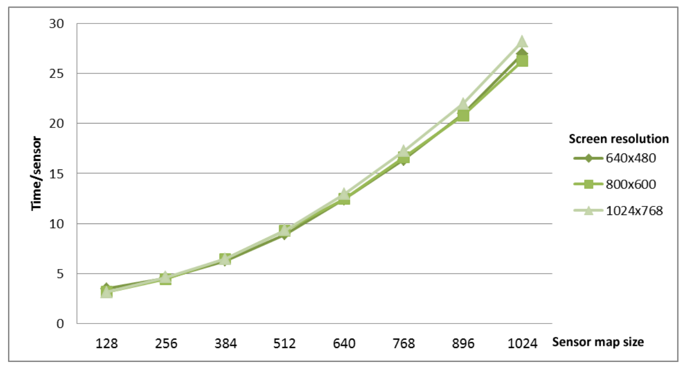

Fig. 10. Second metric: sensor time related to the *sensor map* size

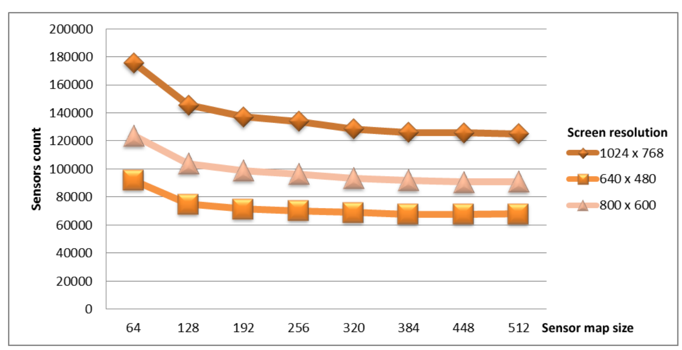

Fig. 11. Third metric: sensors count related to *sensor map* size.

The results were computed using a PC with Intel Core i7 3.50GHz processor with 16GB RAM and NVIDIA GIFORCE GTX 660 GPU, in Windows 7 64 bits.

overly affected.

Some objects of the scene are not good candidate to shade with a global ilumination technique, like the one presented in this work. For example very irregular surfaces, a human face, highly specular surfaces. This type of objects could be excluded of the algorithm and computed with a better suitable technique, substantially reducing the amount of sensors required. Our ilumitation technique allows this kind of flexibility beacause each pixel is totally independent of the rest of the scene.

Some others forms of BRDF could be easily incorporated in the solution replacing the constant factor by a mixed equation. This equation may contemplate diffuse surfaces and some partially glossy mirrors surface. Perfect mirrors would be problematic because the variance of the solution would produce too much noise in the image. Others techniques suit better for this kind of surfaces.

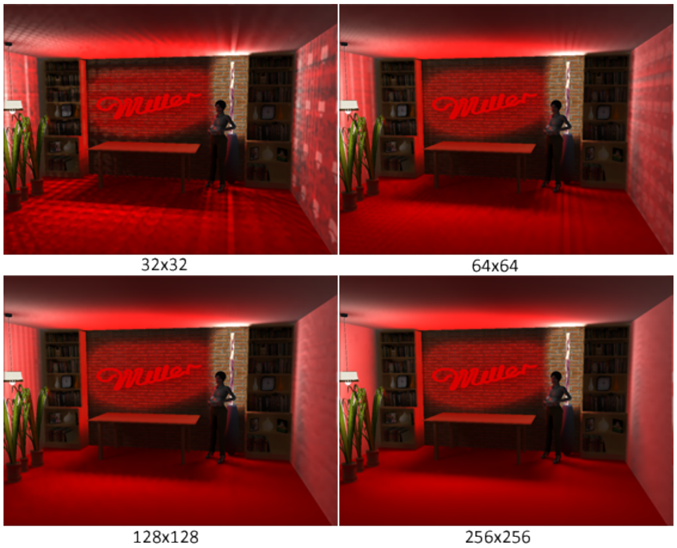

Fig. 12. The sensor area. Left: a 32×32 pixel sensor. Note the noise on the floor and the left wall.

V. Conclusions

The algorithm presented provides an alternate method to solve the global illumination of scene, with support for complex effects like arbitrary-form of light sources, soft shadows, ambient occlusion and color bleeding. The solution is designed to be computed in GPU, taking advantage of many hardware accelerated features. Only one pass is required and it does not need to store any intermediate value, which makes it a highly parallelizable solution. Only a minimal synchronization is required at the end of each tile pass in order to interpolate the values.

The final image obtained by the method has a very realistic looking and it requires relative less processing time than many Ray-Tracing based solutions [17] solutions. Unlike Radiosity it is not required to heavily tessellate the scene, which represents one of the drawbacks of that technique. The original scene data can directly be used by the algorithm.

VI. Future work

The technique is in its early stages. Further studies for optimization are required in order to achieve the performance and scalability of professional rendering solutions.

Its main problems are produced by the limited fov used by the sensor, which avoids the algorithm to capture the illumination from certain angles. Also, point light sources may be difficult to compute since they have to be modeled as a very small surface with a high energy level, making them hard to capture by the sensor. Both problem could be solved by using ray tracing directed to the light sources so as to estimate a statistical value of visibility degree for that light source in the problematic angles. Few rays should be needed so the performance of the solution should not be

Acknowledgment

The authors would like to thank the GIGC Computer Graphics Research Group for supporting this research, and the Department of Information Systems Engineering for providing the support and funding. We also thank Retrovia Project Marta Garcen, Eva Ferrari (English node) for reviewing our work and to the Algebra and Analytic Geometry node to make this contact possible.

References

[1] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis, “Radiometry,” L. B. Wolff, S. A. Shafer, and G. Healey, Eds. USA: Jones and Bartlett Publishers, Inc., 1992, ch. Geometrical considerations and nomenclature for reflectance, pp. 94–145. [Online]. Available: http://dl.acm.org/citation.cfm?id=136913.136929

[2] J. T. Kajiya, “The rendering equation,” in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH ’86. New York, NY, USA: ACM, 1986, pp. 143–150. [Online]. Available: http://doi.acm.org/10.1145/15922.15902

[3] K. Suffern, *Ray Tracing from the Ground Up*. Natick, MA, USA: A. K. Peters, Ltd., 2007.

[4] E. Veach and L. J. Guibas, “Metropolis light transport,” in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH ’97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 65–76. [Online]. Available: http://dx.doi.org/10.1145/258734.258775

[5] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, “Modeling the interaction of light between diffuse surfaces,” in *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH ’84. New York, NY, USA: ACM, 1984, pp. 213–222. [Online]. Available: http://doi.acm.org/10.1145/800031.808601

[6] L. Williams, “Casting curved shadows on curved surfaces,” in *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH ’78. New York, NY, USA: ACM, 1978, pp. 270–274. [Online]. Available: http://doi.acm.org/10.1145/800248.807402

[7] E. Eisemann, M. Schwarz, U. Assarsson, and M. Wimmer, *Real-Time Shadows*. A.K. Peters, 2011. [Online]. Available: http://www.cg.tuwien.ac.at/research/publications/2011/EISEMANN-2011-RTS/

[8] T. Akenine-Moller, T. Moller, and E. Haines, *Real-Time Rendering*, 2nd ed. Natick, MA, USA: A. K. Peters, Ltd., 2002.

[9] A. Keller, “Instant radiosity,” in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH ’97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 49–56. [Online]. Available: http://dx.doi.org/10.1145/258734.258769

[10] M. Pharr and R. Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.

[11] E. Veach, “Robust monte carlo methods for light transport simulation,” Ph.D. dissertation, Stanford, CA, USA, 1998, aAI9837162.

[12] ——, “Robust monte carlo methods for light transport simulation,” Ph.D. dissertation, Stanford, CA, USA, 1998, aAI9837162.

[13] ——, “Robust monte carlo methods for light transport simulation,” Ph.D. dissertation, Stanford, CA, USA, 1998, aAI9837162.

[14] E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec, *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[15] G. Liktor and C. Dachsbacher, “Decoupled deferred shading for hardware rasterization,” in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ser. I3D ’12. New York, NY, USA: ACM, 2012, pp. 143–150. [Online]. Available: http://doi.acm.org/10.1145/2159616.2159640

[16] S. Molnar, “Efficient supersampling antialiasing for high-performance architectures,” DTIC Document, Tech. Rep., 1991.

[17] P. S. Heckbert, “Graphics gems iv,” P. S. Heckbert, Ed. San Diego, CA, USA: Academic Press Professional, Inc., 1994, ch. A minimal ray tracer, pp. 375–381. [Online]. Available: http://dl.acm.org/citation.cfm?id=180895.180928

Fig. 13. Screenshot taken by the Sensor based Lighting technique

Fig. 14. Screenshots taken by the Sensor based Lighting technique