CLASSIFICATION OF GLOBAL ILLUMINATION ALGORITHMS

Hristo Lesev

Abstract. *This article describes and classifies various approaches for solving the global illumination problem. The classification aims to show the similarities between different types of algorithms. We introduce the concept of Light Manager, as a central element and mediator between illumination algorithms in a heterogeneous environment of a graphical system. We present results and analysis of the implementation of the described ideas.*

Keywords: Computer Graphics, Global Illumination, Algorithms, Light Manager

2010 Mathematics Subject Classification: 97R60, 65D18

1. Introduction

The role of global illumination algorithms is to simulate light propagation and interaction in large scale geometry scenes for the porpoise of image synthesis. Global illumination involves the processes of light emission, reflection, redistribution, shadowing and, ultimately, absorption in an environment. These are physical processes governed by the equations of radiative transfer. These equations are based on first and second laws of thermodynamics which describes how thermal energy is conserved and flows from regions of high potential to regions with low potential [1]. This process can be described using geometrical optics formalism and physical and wave optics effects can be restricted to the level of scattering and emission at surfaces. Given the forgoing physical assumptions we can specify an equation for global illumination.

Let M denote the collection of all surfaces in an environment. Let X be a space of real-valued functions defined on $M \times S^2$, that is, over all surface points and angular directions in the unit sphere S^2. Given the surface emission function $g \in X$, which specifies the origin and directional distribution of emitted light, we wish to determine the surface radiance function $I \in X$ that satisfies

$$I(x, x') = g(x, x') \left[\varepsilon(x, x') + \int_{\Omega} \rho(x, x', x'') I(x, x'') \, dx'' \right]$$

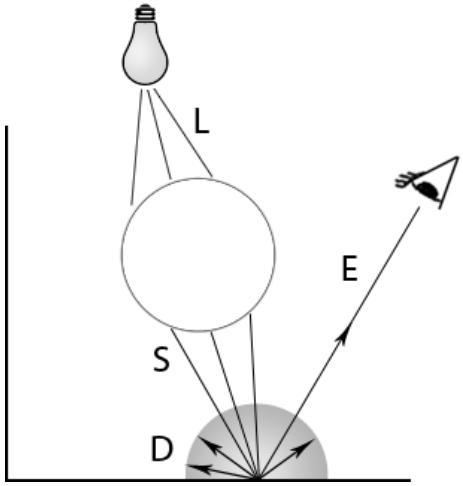
where Ω is the hemisphere of incoming directions, ρ is a directional reflectivity and x'' is a point on a distant surface determined by x and x'. This equation was introduced by Kajiya [2] and describes how light is propagated through a scene, in terms of the physical principles discussed above. Virtually all modern

photorealistic rendering architectures are based on this integral equation also known as the rendering equation [3]. Examined more carefully it becomes clear that this recursive equation has no analytical solution (except in some relatively simple cases): to overcome this fact, rendering software must use approximations to produce visually plausible solutions and this allows the use of many different algorithms.

2. Light path notation

When describing a light path it is often necessary to distinguish between different types of surface reflections along the path. Heckbert [4] has introduced a compact notation for exactly this purpose. Heckbert's notation classifies different light paths by vertices in the path and type of event that caused creation of the vertex. The notation has four types of vertices:

- L - a light source
- E - the eye
- S - a specular reflection
- D - a diffuse reflection



If we want to describe a combination of paths, regular expressions can be used. As an example L(SD)+DE means a path starting at the light source having one or more diffuse or specular reflections before being reflected at a diffuse surface towards the eye.

A global illumination algorithm is expected to model all types of light paths, that is, it must have L [D|S]*E type.

3. Global illumination algorithms classification

We can classify different global illumination algorithms by the approaches each of them is taking to solve the rendering equation [5]. This gives us several categories:

- **Exact - Approximate**: depending from using unbiased or biased approach in solving the equation and how they reduce computational errors;
- **Gathering - Shooting**: depending from how they track light paths direction;
- **View dependent - View independent**;
- **Hybrid**: combination from other approaches.

3.1. Exact

Despite the fact that the rendering equation has no exact solution there are numerical methods that can minimize computational errors and produce result very close to what is expected. Other way to name them is Brute Force methods, because they attack the rendering equation directly and try to solve it iteratively, which consumes too much time and resources working on relatively complex scenes. On the other hand, because the fact they are solving the equation as a whole, the exact methods can simulate every aspects of the light path without need for any modification. Mathematical core of the exact methods is the Monte Carlo method for solving integral equations. In real life situations the time given for achieving solution is limited, so when the computation is done there is always noise introduced in the final result. This noise represents the relative error in the current algorithm iteration and will be much lower in the next iteration, expected to eventually disappear after infinitely long time. This fact allows us to call these methods also “unbiased methods”. From the user’s point of view algorithms based on exact computational methods are very easy to work with, because of the relatively small numbers of parameters they depend on and their relatively small memory footprint. One big disadvantage is that these methods are not adaptive, so they cannot concentrate very well on specific parts of the rendering equation and in small time frames they are very noisy. Some light path scenarios are still problematic, like light coming from the Omni light source.

Some well-known exact methods are: Path tracing, Bi-directional path tracing, Metropolis light transport [6].

3.2. Approximate

These algorithms use adaptive approach in solving the rendering equation and concentrate resources in those parts of the equation which are important for the final image. This fact makes them faster than the exact methods and allows them to deal in a much easier ways with some light paths that are considered difficult for the other methods. Approximate algorithms often rely on some visual metric to tell if the result is accurate enough, so they can finish working. Another advantage is that they can be interrupted at any time and their result can be cached and used later to finish the computation without the need to restart work. Because of their approximate nature these methods introduce computational errors in the final result which makes them biased and physically incorrect. Because of these errors there can be visual artifacts in the final image caused by scene configurations that don’t match the algorithm's visual metric. From user’s point of view these methods are much more complicated to use because of the many parameters that control the

visual metric. Some well-known approximate methods are: Photon mapping [6], Irradiance caching.

3.3. Shooting

Shooting algorithms trace a light path starting with vertex L originating from the light source and aim to finish it with vertex E. Some effects like caustics are better reproduced using shooting algorithms. One advantage of these algorithms is that they follow the natural flow of the light energy through space. This can also be a disadvantage, because often time and resources will be spared to trace the light path in some scene places that are not be visible from the current point of view and so won't have any contribution for the final result. Another disadvantage is that because the light is shot in the scene as photons with infinitely small radius, scene regions that are far from the light source will be computed with insufficient precision, and will need more photons to be shot at them. This is why shooting algorithms are rarely used to simulate natural light coming from distant objects like the sun and nonphysical light sources. Some well-known shooting methods are: Photon mapping, Light tracing.

3.4. Gathering

Gathering algorithms trace light path in the opposite way of shooting ones. Gathering algorithms starts tracing light paths with vertex E originating in the eye and tries to finish the path with vertex L in the light source. Thus these algorithms spend much work on these parts of the scene which are visible. This makes them more efficient than shooting methods, because every light path they trace will be part of the final result. Gathering algorithms can simulate very complex light sources and scene geometry. Some types of light sources are difficult to capture using gathering algorithms. For example, if the light source is very small, it is likely to be overlooked by most of the traced rays and this will introduce noise in the final image. Some well-known gathering methods are: Irradiance caching, Path tracing.

3.5. View-dependent

These algorithms consider scene surfaces visible from the eye only. These surfaces can be directly visible or seen through secondary (indirect) reflected vertex in the light path. So view dependent algorithms spend more time working on details that will be visible and will have most impact over the final image. Advantage of these algorithms is that they don't impose any restrictions over scene geometry representation. Caching is often used to facilitate the work of the view dependent algorithms. These methods can be adapted to sample more heavily these parts of the rendering equation which are important for the specific scene and view point. Their main disadvantage is that when the view changes all computational work must be started all over again. Some well-known view-dependent algorithms are: Irradiance caching, Path tracing.

3.6. View-independent

These algorithms calculate light energy flow through the entire scene just once. The time spend doing these pre-calculations can be very long and can consume lot of resources but once the work is done the viewer can move freely

through the scene very fast using cached light information. This is very useful in real-time graphics applications like games and architectural presentations. Some disadvantages of view-dependent algorithms are that they must compute light energy in all of the scene regions without knowing if every region will be visited ever. Usually these algorithms have special requirements for the geometry representation of the scene because they are using some kind of finite elements analysis over the scene geometry to calculate the light flow. View independent algorithms cannot cache some light paths that depends on the viewer's position like specular reflections and these have to be calculated for every new frame of the simulation. In general, only the diffuse term can be captured by these algorithms. Some well-known view dependent algorithms are: Radiosity, View-independent Irradiance Map [7].

3.7. Hybrid

Hybrid algorithms are combinations of previously described algorithms. These combinations are aimed at removing a specific weakness of a specific algorithm using ideas and approaches from other GI algorithms. The two algorithms can communicate through their results and find faster and better solution. Hybrid algorithms usually work at several passes: during the first pass one algorithm calculates one part of the light paths and caches the results, on the next pass the second algorithm computes other types of light paths and uses these cached results to produce the final image. A disadvantage of hybrid algorithms is that these combinations between different algorithms are very concrete and usually the two algorithms have to be redesigned to work with each other. Some well-known hybrid algorithms are: Photon Map + Irradiance caching (Final gathering) [6], Radiosity + Irradiance caching.

4. Light manager

As mentioned earlier, the main disadvantage of the modern rendering system is the strong connection between different components of the system. To eliminate this flaw in rendering systems design this paper will introduce the concept of Light Manager. Light Manager is central component of the system which provides a weak connectivity for other parts of the rendering system.

Most important step in the process for every illumination algorithm is to generate new ray to trace light energy. Next important thing is to determine the light path type this ray belongs. In classic rendering systems architecture this is function of the shading modules which contains information about the surface visual properties. Here this function is removed from the shaders and is given to the Light Manager.

The main reason for this is to allow different global illumination algorithms to work together seamlessly on any part of the rendering equation that can handle best, without having to know the other algorithms.

When the new algorithm is added to the system the Light Manager should be informed about which light paths this algorithm can work on. Here under the term

light paths we should not only understand the Heckbert's notation, but also the sequence of the current ray in the context of the entire light path.

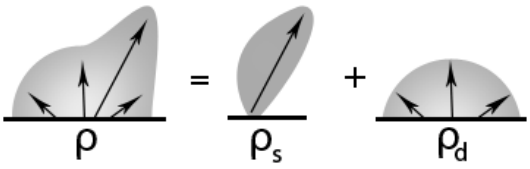
When the system needs to calculate the indirect light reaching some surface point Light Manager should be called to do the task. If some algorithm (like Photon mapping and Irradiance Cache) has already cached the indirect illumination Light Manager should ask that algorithm to approximate the illumination around this surface point using stored illumination data.

If only a brute force algorithm is available the Light Manager should ask it to generate new ray and trace it through the scene. This approach guarantees that every illumination algorithm should communicate only with the Light Manager. This ensures that we can integrate new illumination algorithms in the rendering system without the need to know specific details about other illumination algorithms that are already part of the system. This architecture is driven by the plug-ins representing different illumination algorithms and allows the user to make unlimited configurations for how the scene will be illuminated.

Although the algorithms cannot communicate directly with one another, there are cases in which the order of algorithm execution is highly specific. To avoid any conflicts, when instantiated, Light Manager should build a graph of dependencies between algorithms that are registered in the system. Using this graph Light Manager can call them in the correct order.

The mathematical idea behind Light Manager is to divide the rendering equation integral into parts which represent direct light, indirect light and reflections. Let's have a closer look at how this division can be done as shown by Jensen [6].

First, the reflectivity function ρ can be represented as a sum of two simpler functions: specular reflection ρ_S and diffuse reflection ρ_D.



$$\rho(x,x',x'') = \rho_S(x,x',x'') + \rho_D(x,x',x'')$$

This process separates high frequency signals (specular reflection) from low frequency diffuse reflections and gives more control over the noise in the final image. We can also represent the rendering equation as sum of all light paths arriving at surface point x. But first we should rewrite the rendering equation in hemispherical form, where Ω will be the hemisphere area.

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_{\Omega} \rho(x,\vec{\omega'},\vec{\omega})\, L_i(x,\vec{\omega'})(\vec{\omega'}\cdot\vec{n})d\vec{\omega}$$

Now it is easier to write the sum equation.

$$L_i(x,\vec{\omega}) = L_{i,l}(x,\vec{\omega}) + L_{i,c}(x,\vec{\omega}) + L_{i,d}(x,\vec{\omega})$$

Here $L_i\left(x, \overleftarrow{\omega}\right)$ represents incoming light in point x, $L_{i,l}\left(x, \overleftarrow{\omega}\right)$ is the light path with only one vertex of type L, which hit directly x, $L_{i,c}\left(x, \overleftarrow{\omega}\right)$ is the light path with one or more vertices of type S that reached x and finally $L_{i,d}\left(x, \overleftarrow{\omega}\right)$ represents indirect illumination in x as light path with one or more vertices of type D.

Finally we can represent the rendering equation as a sum of four integrals each of which represents different type of light path.

$$\int_{\Omega} \rho(x, \overrightarrow{\omega'}, \vec{\omega})\, L_i(x, \vec{\omega'})(\vec{\omega'} \cdot \vec{n}) d\vec{\omega}$$

$$= \int_{\Omega} \rho(x, \overrightarrow{\omega'}, \vec{\omega})\, L_{i,l}(x, \vec{\omega'})(\vec{\omega'} \cdot \vec{n}) d\vec{\omega} + \int_{\Omega} \rho_S(x, \overrightarrow{\omega'}, \vec{\omega})\, (L_{i,c}(x, \vec{\omega'})$$

$$+ L_{i,d}(x, \vec{\omega'}))\, (\vec{\omega'} \cdot \vec{n}) d\vec{\omega} + \int_{\Omega} \rho_D(x, \overrightarrow{\omega'}, \vec{\omega})\, L_{i,c}(x, \vec{\omega'})(\vec{\omega'} \cdot \vec{n}) d\vec{\omega}$$

$$+ \int_{\Omega} \rho_D(x, \overrightarrow{\omega'}, \vec{\omega})\, L_{i,d}(x, \vec{\omega'})(\vec{\omega'} \cdot \vec{n}) d\vec{\omega}$$

Light Manager provides explicit interface for the algorithms, so programmers can specify which part of this equation their algorithm will try to solve. Another task for it is keeping the sum calculations in the right order and switching between algorithms. Another advantage of the Light Manager is that it can be used for further optimization of the rendering process based on mathematical domain representation.

5. Analysis and results

The ideas described in this paper are implemented by the author in the rendering system “RayTracer” [8] which is used in training students studying Informatics at the “Paisii Hilendarski” University of Plovdiv. The system is developed in C# and uses .Net 2.0 framework. Several global illumination algorithms were implemented using the Light Manager paradigm and tested in scenes with heavy geometry and number of light sources. The list of implemented algorithms includes: Path Tracing, Irradiance Caching, Photon Mapping, and View-independent Irradiance Map. Through numerous tests Light Manager based architecture of the “RayTracer” has proved to be easily extendable with plug-ins and also facilitates the process of transformation of existing global illumination algorithms to be used as part of the system.

6. Conclusion

We have presented detailed classification of the existing types of algorithms which solves the global illumination problem in realistic image generation process. The new concept of Light Manager and its uses was introduced. This concept was justified mathematically and also from programmer’s point of view. The results discussed above support the ideas described in this article. For the future, there is

much work to do, using this implementation of the Light Manager will be very useful to implement an optimization core around it which will have great impact in the computer graphics domain.

References

[1] M. Planck, “The Theory of Heat Radiation”, New York, Dover Publications, (1988)
[2] J. Kajiya, "The rendering equation", Siggraph Volume XX, Issue IV, (1986), pp. 143 – 150
[3] P. Dutre, K. Bala, P. Bekaert, “Advanced Global Illumination Second Edition”, Wellesley Massachusetts, A K Peters, (2006)
[4] P. Heckbert, “Adaptive radiosity textures for bidirectional ray tracing”, ACM SIGGRAPH ’90 Proceedings, (1990), pp. 145–154
[5] S. Laszlo, “MONTE-CARLO METHODS IN GLOBAL ILLUMINATION”, Vienna, Institute of Computer Graphics Vienna University of Technology, (2000)
[6] H. W. Jensen, “Realistic Image Synthesis Using Photon Mapping”, Natick Massachusetts, A K Peters, (2001)
[7] H. Lesev, D. Ivanov, “VIEW-INDEPENDENT IRRADIANCE MAP GENERATION”, Burgas Free University Almanac Vol. XVI, (2007), pp. 250-256
[8] H. Lesev, “PHOTOREALISTIC COMPUTER GRAPHICS IN INFORMATICS EDUCATION PROCESS”, Conference Education in the Information Society, (2010), pp. 141-146

Hristo Iliyanov Lesev, PhD student
Faculty of Mathematics and Informatics
University of Plovdiv
236 Bulgaria Blvd.
4003 Plovdiv, Bulgaria
e-mail: hristo.lesev@gmail.com