Light octree: global illumination fast reconstruction and realtime navigation
Vincent Vivanloc, Jean-Christophe Hoelt, Coong Binh, Mathias Paulin

To cite this version:

HAL Id: hal-01518549
https://hal.science/hal-01518549
Submitted on 9 May 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract

We present a method to rapidly build an irradiance cache based on a local illumination environment approach. This cache is obtained by a stream simplification of a photon map. The photons are K-Means clustered per voxel into sets of virtual directional light. These lights are stored into an irradiance texture to provide a real-time rendering of a global illuminated scene. This method can be integrated into an existing GPU shader to obtain complex material rendering and can be accelerated by texture atlases.

Keywords  global illumination, local illumination environment, irradiance cache, octree, real-time

1 INTRODUCTION

The overall quality of computer rendered scene can be greatly enhanced by taking into account the indirect illumination [TL04]. In one hand, the classical OpenGL direct lighting pipeline can be improved by multipass techniques for real time effects as shadows, specular reflections, refractions [SKALP05] and splat based caustics. On the other hand, the rendering of a lower frequency lighting is provided by precalculated light maps based on radiance transfers. These maps are often computationally expensive to generate and limited to diffuse radiosity.

We develop a method to generate these maps rapidly from photon maps by a simple and parallelizable algorithm. Our simplification relies on local illumination environment (LIE [FBG02]) approach. The scene space is divided into voxels filled with a compact representation (Virtual Directional Light) of the irradiance. Since the process is done locally, the simplification error is minimized compared to a global scene scheme.

Our representation of illumination could be used as an irradiance cache for ray tracing. However, such expensive rendering must be reprocessed at every viewpoint change. Since we would like a real-time rendering of global illumination solution, we develop a viewpoint independent rendering method, based on a GPU rendered octree. The sets of VDLs are directly stored in the GPU memory and can be integrated with advanced material shaders. Eventually, we propose a render-to-atlas procedure to improve the framerate and enable rendering on legacy OpenGL hardware.

Our contribution is:

- a fast and stream simplification of a photon map;
- a real-time rendering of indirect illumination on a GPU without scene remeshing
  - integrable in existing shader pipelines for advanced material rendering
  - running on legacy OpenGL hardware (requires a scene parametrization)

2 PREVIOUS WORKS

2.1 Creating an illumination cache

A global illumination solution can be provided in screen space, using path tracing [Kaj86, LW93] or better, in scene space, by radiosity [GTG84] or photon mapping [Jen96, Jen01]. Contrary to the former
screen space method, a scene space solution can be reused when the scene viewpoint change. We choose the photon map approach since it takes into account not only caustics but also directional diffuse and specular indirect irradiances.

2.1.1 Cache sample representation
A radiance cache is a scalar field of a reflectance function values. Such function links the incoming irradiances to the radiance. For both flexibility and efficiency, the irradiance is preferred to the radiance. The convolution between radiation and the 5 dimension reflectance function can produce a high memory consuming vector field. Such field must then be compactly stored in a continuous or a discrete way.

A continuous representation consists in fitting a discrete set of photons into a set of coefficients bound to a function basis, such as spherical harmonics (SH) or wavelets. SHs were the first used for precomputed radiation transfer (PRT) [SKS02], for diffuse [RH01] and glossy [LSSS04, KGPB05] materials. Contrary to wavelets, SHs are not directly applicable to high frequency materials. Recently, wavelet coded BRDFs have been rendered on GPU [WTL06]. However, both of these continuous representations require heavy precomputations, for instance a clustered PCA analysis [SHHS03] for SHs.

In a discrete representation, the photon map can be considered as a set of discrete lights: point light (Virtual Point Light [Kel97]) or directional light (Light Vector) [ZSP98]. Such discrete representations of lights are grouped by clusters which similarity distances are based on density [CLSS97], visibility, power, position [SWZ96, PPD98] or perception values [FBG02, WFA°05]. Even if discrete representations are more prone to aliasing than their continuous counterparts, their simplicity is well suited with a stream processing of the photon map.

2.1.2 Global and local cache
The transformation of a set of photons into an irradiance representation can be done on the whole scene, resulting in a global cache. The simplification bias can be reduced by working locally in limited regions of space, leading to a local cache. These space partitions can be constructed on a per scene object basis and more generally, with a regular grid, or a less memory consuming octree. Such structure have been used for renders with massive number of lights , assuming that such lights have only a local influence on the environment [FBG02]. The octree can be built on geometric criterion or subdivided using an irradiance threshold, to create an irradiance volume [GSHG98, PH04].

2.2 Rendering an illumination cache
A fast reconstruction of a global illumination solution is an open problem. Rendering directly the photon map needs a huge number of photons to be casted. Hybrid methods can reduce the number of casted photons: high frequency effects like shadows or specular reflections/refractions are processed by GPU rasterisation whereas photon mapping is reserved for lower frequency irradiances [LC04].

2.2.1 Global cache rendering
The reconstruction of a photon map can be done in screen, texture or object space.

The screen space reconstruction always offers the highest lighting quality since it is a per pixel computation. The final gathering pass is the most time consuming part of rendering [Jen96]. It has been parallelized on CPU [WKBT02] and GPU [Hac05], but must be reprocessed on each camera move.

On the contrary, an object space reconstruction is viewpoint independent: the cache samples are bound to the mesh vertices. PRT renderings are often applied on high tessellated, static and single object scenes [SHHS03]. On the other hand, the scene could be dynamically re-meshed to match the illumination distribution with its geometry [WHSG97]. Such per vertex lighting could increase dramatically the geometric complexity of the scene.

A texture space reconstruction is also independent from the camera position [Arv86, Shi90]. For instance, a static or dynamic [Nie00, NC02] precomputed radiosity can be bound into a simple 2D texture. Otherwise, hybrid object_texture space reconstruction of the photon map has been done on a hybrid CPU/GPU hemicube final gathering but still require a well tessellated scene [LC04].

The photon splatting method can be affected to this texture space reconstruction category. This method is an application of point based rendering. The final gathering is replaced by its dual operation: instead of fetching the contribution of the neighbouring photons to estimate a photon density, this density is summarized by a radial function. Practically, a gaussian centered on photon is directly splatted on screen. It has been applied to diffuse [LP03] and glossy materials [GKP05]. Since photon splatting adds geometry and fills the scene, real time framers are only reached for low frequency indirect lighting [DS05] or space constrained caustics [SKALP05, KBW06, WD06].

2.2.2 Local cache rendering
The previous methods can also be applied to local irradiance caches, bound to localized regions. Like their global counterparts, such caches can be generally used by screen space rendering [CB04] to accelerate the final gathering stage. Otherwise, GPU rasterisation techniques are combined to render such caches in real-time. Precalculated radiosity can be linked to a sphere map [WTP00] or a cube map [NPG03]. These methods are limited by cube map constraints (convex objects and concave environment).

Eventually, the rendering can also be done per voxel. The first approach used a multi-pipe SGI GPU with well known clustered OpenGL lights [UH99]. The octree texture is a more affordable solution, running on
commodity hardware. Such octree textures were proposed for a multiresolution and intuitive object volumetric painting. They do not require any texture surface parametrization [gDGPR02, BD02] and can be run on GPU [ILHNO5, KLS*05]. Such methods can also be derived for animated lights. The light source clouds are clustered and compressed into SH to be finally bound to voxels [KAMJO5]. However, such process requires hours of precomputations.

3 GENERAL ARCHITECTURE

The architecture of our rendering pipeline is based on a hybrid renderer, assuming that the illumination can be separated into direct and indirect components. Stochastic indirect illumination methods can hardly represent high frequency signals even with a large amount of photons. Practically, shadows, specular reflections or refractions are less detailed. Thus, these high frequency effects must be rendered by a different pipeline.

In this paper we focus on indirect illumination rendering pipeline. This pipeline is composed of 3 main stages (figure 1). The geometric preprocess stage contains the octree creation. The photon map clustering stage, processed on the CPU, is a highly streamable process, described on the section 3.1. Finally, the GPU rendering provides a real-time rendering of indirect illumination, detailed in the rendering section 3.2.

3.1 Photon map clustering

3.1.1 Prefiltering

A photon map with less than one million photons is often too much biased to be directly used. To reduce this bias, the photon map is clustered per voxel. However, such a histogram approach assumes estimation areas are equivalent [Jen01]. This is roughly true for a high resolution octree and a very tesselated geometry. In practice, this is false for coarse octree or for a geometry made of large planes. In such configuration, the photon density can be underestimated since the probability to hit the voxel/polygon section can be very low. Visually, it is translated into bands of different colors artefacts (see figure 16-a). Therefore, a KNN filtering is compulsory to smooth the photon density across the voxels.

KNN parametrisation

Such filtering requires two parameters, k, the number of nearest neighbors and r, the radius of region of gathering. In our case, these parameters are set arbitrary 5. The optimal (k,r) values are found experimentally to reach an acceptable visual aspect in a reduced preprocessing time. Concerning the gathering region, we have chosen an ellipsoid aligned on voxel normal. This could induce some artefacts explained in 5.3.1.

3.1.2 Clustering

In order to provide a GPU friendly representation of the photon map, the illumination cache sample must be compacted in order to fit in a tiny GPU memory. We conceived a simple method to cluster the photons into virtual directional lights.

Choice of clustering method

The principal component analysis (PCA) and its derivative (Clustered PCA [SHHS03]) can be very time consuming. Hierarchical clustering (HClust) produces a clustering at all levels and returns an unique solution, but is in $O(n^2)$, where n is the number of samples. KMeans [Mac67] cannot produce a unique solution but requires a lower memory footprint. Its theoretical worst-case complexity is polynomial [HPS05, AV06]; however, in practice the number of Kmeans iterations required to obtain a solution, is low. In our case, for a given population of 50 to 500 photons per voxel, with the simplifications described below, the total time spent to create up to 8 clusters is below 1 second to 1 minutes depending on number of voxels.

KMeans parametrisation

As automatic a classification algorithm could be, we need to define at least two parameters: the sample to cluster-centroid distance function and the resulting entity after the fusion of cluster samples.

An accurate definition of a similarity between two photons would consider the 5 dimensions of the irradianc. This would be too computationally expensive, so we assume the voxel volume is little enough to discard the photons position. Consequently, the distance is defined as the cord length formed by the two photon incoming directions.

The first fusion entity considered was an accumulation of the irradiance. However, this simplification has a great visual impact on the rendering (see figure 2). That is why we introduce the virtual directional light (VDL), a (irradiance; incident direction) pair. The VDL can be compared with the light vector
1. Dependant complex material (figures)

2. Therefore, our irradiance texture can be easily integrated to an existing shader to render surface-determined shaders (figure 12).

3. The light vector represents a directional radiance whereas the VDL contains only an irradiance and lets the GPU process the radiance. Contrary to HClust, KMeans works on a given number of clusters. Since we are limited by the GPU memory and shader sizes, we fixed the maximum number of VDLs to 8.

4. Photon map rendering

5. Our clustered photon cache could be used in a classical screen space rendering, but we applied it to GPU for real-time rendering. The octree rendering with directional light rendering, contrary to photon splatting, is less sensitive to local geometric variation, since the irradiance is implicitly bound to the surfaces intersecting the voxel (figure 2).

6. The set of directional lights is bound to each voxel and can be integrated by shaders as classical OpenGL directional lights to process illumination locally. It takes into account the variation of the surface and thus, enables the rendering of surface-dependent material. Therefore, our irradiance texture can be easily integrated to an existing shader to render surface-dependent complex material (figures 2,11).

7. The voxel volume is assumed to be little enough to ignore occlusions. Practically, VDLs do not cast any shadows. However, parallax occlusion mapping algorithms [PO06, Tat06] could be integrated to our rendering pipeline to improve the quality of surface-determined shaders (figure 12).

8. IMPLEMENTATION

9. Our pipeline is running on a dual bi-core 2Ghz AMD64 Opteron 270’s with NVidia GeForce 4500 GPU. It is compiled under Linux with gcc 4.0.2 and shaders are implemented in Cg under a fp40 profile.

10. Photon casting and clustering

11. The photon map is loaded from a modified version of Yafray [WEEdG’06], a modular global illumination renderer. It has been chosen for its fast photon casting[1] and its tight integration with Blender, the well-known powerful 3D modeler and renderer.

12. The octree is built on the whole scene, using a geometric subdivision criterion. An octree built on light irradiance density must be rebuilt at each new photon arrival and prevents any octree GPU filtering.

13. We attempt obtaining an efficient KNN filtering. We replaced the photon gathering by a scattering, implemented by screen and space octree splattings. For screen space splatting, we got a blurred rendering with low performances, induced by some fillrate overheads. For space splatting, we suffered from memory shortage due to a compulsory storage of the splatted photons. We also tried a GPGPU implementation but such scattering algorithm does not suit to existing GPU architectures. We fetched photons directly using the octree. Unfortunately, compared to a KD-tree, it almost doubled the photon gathering time. Finally, we came back to a classical KD-tree KNN gathering.

1. few minutes for a 1 million photon map

Figure 2: Color and light octree photon map simplifications. In the left picture, the scene seems to be foggy, due to the trilinear filtering and to the loss of irradiance directional information. In the right picture, the ball is clearly illuminated by two distinct lights, thanks to the VDLs. OpenGL rendering is provided for comparison.

Figure 3: Octree clustering

Figure 4: Photon map preprocessing on CPU
4.2 Photon rendering

4.2.1 Enhanced GPU octree

The light octree is an extension of the GPU octree texture provided by [LHN05]. The octree is entirely coded in a texture stack implemented by a 3D texture. A flag in the texel alpha channel indicates if the RGB field is an address to the next octree sub-node or a leaf. For an octree of depth δ, a maximum of δ texel accesses is done. The octree texel are in 8 bit integer format because float 3D textures are not available on our hardware configuration. Since we do need a float storage, we introduce a secondary texture containing irradiance data (figure 5). The octree leaves contains the addresses to the data texels. Since actual textures are limited to a 4096x4096 resolution, a 24bit coding for these addresses is sufficient (see figure 7). A texture memory manager reduces the original memory footprint from 64MiB to 512KiB(octree)+4MiB(VDLs), for 100k leaves of a δ = 6 octree with k = 4 VDLs.

Since the color is constant per voxel, the result has a blocky aspect. This can be alleviate with an interpolation. This per pixel smoothing is done by a brute force trilinear interpolation between 8 neighbour voxels.

4.2.2 VDL rendering

We consider a virtual directional light \( VDL(c,d) \) where \( c \) is the irradiance converted in RGB color and \( d \) its direction. Storing the color and direction of the virtual lights in two texels wastes the two alpha channels and lowers the performances due to the additional texel access. The virtual light is packed into one unique texel by compressing the direction into a single float in the alpha channel. The direction is converted from a cartesian \((x,y,z)\) to a spherical \((\theta,\phi)\) frame\(^3\) and the angles are discretized into 8+8 bits values (figure 8).

Once retrieved from the irradiance texture, the VDLs can be integrated easily within any illumination shader.

\[
c_{VDLs} = \sum_{i=1}^{n_{VDLs}} brfd(d_i, v, n) c_i
\]

where \( c_i, d_i \) are resp. the irradiance and the direction of the \( i \)th VLD, \( n_{VDLs} \) is the number of VDLs, \( v \) the view direction and \( n \) is the surface normal.

\(^2\) A bicubic filter would require 24 samples which is too expensive.

\(^3\) Within a normalized frame, \( r = 1 \)
some specular materials have been simulated using shaders to simulate complex material. For instance, the light octree can be integrated to surface bound packed into a 2D texture (figure 9).

4.2.3 Render to atlas

For a light octree, a single fragment requires \( n_{DL} + \delta + 1 \) texture accesses. The trilinear filtering can worsen the performances by multiplying accesses by 8. The number of texture fetch can be drastically reduced to a single access using an atlas light map. Such process has been done for radiosity [RUCL03], we extended it to our light octree.

The scene objects must be parametrised into atlases before any rendering. This can be done manually with Blender or automatically using our geometric preprocessor. The charts are built using a Voronoï construction method [SWG+03]. Each chart is then parametrised into atlas using the Least Square Conformance Map [LPRM02] algorithm and is finally tightly packed into a 2D texture (figure 9).

Once the scene is parametrized, the render to atlas is done on the fly by rendering the 3D octree in the atlas, capturing it and applying the resulting texture on the model (figure 10).

5 RESULTS

5.1 Visual aspect

The light octree can be integrated to surface bound shaders to simulate complex material. For instance, some specular materials have been simulated using simple cube map and parallax bump map shaders. Nothing but the shader complexity, would prevent to integrate the irradiance texture into more realistic shaders (figure 12).

The visual differences between texture atlas rendering and its light octree counterpart mainly come from the more of less visible atlas seams. An integer texture format could also clamp down the color dynamic (figure 10).

Contrary to PRT techniques based vertex sampling, the surface irradiance distribution is split per voxel. The octree is therefore more robust to the high variations of the irradiance, which can be seen on caustics close up of the figure 11.

5.2 Timing results

The CPU preprocessing time is linear in term of voxels (table 13). The geometry complexity has a lesser importance. The compression time spent for clustering is low (few seconds) regarding the KNN gathering (figure 17). The less photons are gathered, the faster the computation is processed (table 14). This stage is the most expensive part of the compression scheme but cannot be skipped, as explained in the subsection 3.1.1.

A progressive updating process allows the user to rapidly preview a globally illuminated scene. The octree is updated progressively from coarse to fine depth octree. A rough visualisation (\( \delta = 4 \)) is available in less than 10 seconds and an acceptable result (\( \delta = 5 \)) is achieved in less than 1 minute (on figure 13).
On one hand, we cannot compete qualitatively with Yafray offline global illumination renders. The screen space approach gives visually better results because the underlying coarse irradiance cache is compensated by a per pixel final gathering. On the other hand, our solution provides a real time navigation in a roughly equivalent scene (figure 15).

In our method, a compression error is directly translated into visual artifacts. The purpose of this subsection is to provide some solutions for such rendering errors.

5.3.1 Octree aliasing

Our method may suffer from three sampling artefacts that should be resolved by additional per voxel computations.

First, the density underestimation, visible on large planes (figure 16-a), can be alleviated using KNN filtering (subsection 3.1.1).

However, this filtering is also the cause a second artefact, the energy bleeding on object edges and corners (figure 16-b). In a screen based final gathering,
the gathering space is constrained by a cone built using the photon impact normal. In our gathering, we define an ellipsoid oriented along a voxel normal. Per definition, a voxel has no normal. In the current version, it is approximated as an average amongst the intersecting geometry.

Third, visual banding artifacts (figure 16-c) can be produced by the void voxels fetched by the trilinear interpolation. This may be prevented by virtually extending the voxel bounds. Unfortunately, a too large overlapping zone can produce visible defects.

Artificially expanding the voxel boundaries can also impede the photon density estimation: the photon gathering is more time consuming and less accurate. In summary, a careful balance must be set between a blurred picture and a better but slower result.

5.3.2 Number of VDLs
The second source of error is bound to the clustering itself. The more the clusters are, the better the results would be, even if the framerate would slightly decrease. The intra cluster error is measured in term of standard deviation between the direction of cluster centroid and the directions of the clustered photons, expressed in cord length unit. This angular error is easily explained by the low number of clusters: a hemisphere ($180^\circ$) is partitioned between only $k$ clusters.

For $k = 4$, the standard deviation around its cluster centroid ($\pm 20$ degrees) seems to be high but the overall quality of picture is preserved. For lowers $k$, the higher error rate induces visible artefacts (figure 17).

5.3.3 Voxel size
The voxel size often matters. If it is too large, the photon clustering is a low-pass filter and the irradiance is blurred. Otherwise, if it is too little, too few photons are intersected, leading to a lighting underestimation. Experimentally, the quality improves for $\delta = [4, 6]$. A depth $\delta = 6$ brings the octree resolution to 64x64x64. For a given scene contained in $1m^3$, a voxel has a volume of $1/64 \approx 1.5cm^3$, which could be low in term of accuracy but rather sufficient to improve the visual aspect against a classic OpenGL rendering. For $\delta = [7, 8]$, the effort spent to smooth the density does not provide significant improvements.

6 CONCLUSION AND FUTURE WORKS
The photon map clustering is an efficient technique to rapidly produce an irradiance cache. Compared to radiosity maps, the light octree provides more visually appealing scenes thanks to a photon map. Our sets of directional lights localized per voxel reduces the global compression error and is highly integrable to existing GPU shaders.

Techniques to incrementally update our octree should be examined. Many Kmeans algorithm incremental...
versions have been proposed. Eventually, a continuous representation, with SH or wavelets should be also compared with our discrete VDL representation.

ACKNOWLEDGMENTS

This research is performed in the frame of the RNTL Project Lumière, funded by the French Board of Research involving IRIT/SIRV, EADS/CCR laboratories, Renault, EADS companies and OKTAL IT service company. Models courtesy resp. by the Stanford 3D Scanning Repository, Fabrice HEILLOUIS (DMI cars) and Renault Centre Technique de Simulation.

More information on http://www.irit.fr/lumiere

REFERENCES


MacQueen. Some methods for classification and analysis of multivariate observations. 5th Berkeley Symposium, 2:281–297, 1967. 3


