ハードウェアの支援による
高速な大域照明レンダラーの開発

Global Illumination using Programmable Graphic Hardware

蜂須賀　恵也
Toshiya HACHISUKA

東京大学　工学部　システム創成学科

ABSTRACT. Global Illumination is essential to photorealistic image synthesis, but problem is that it is very hard to use in practical due to its large computational time. Recently, programmable graphic hardware comes into available and we can program various processing on graphic hardware. Programmable graphic hardware can be considered as powerful parallel numerical processing unit, so it is possible to accelerate some numerical calculation. The developed algorithm can render photorealistic image relatively fast by exploiting the power of graphic hardware.

１．背景

大域照明は光と物体との相互作用をすべて考慮し、写実的な画像を得るための計算モデルの事であり、ここ数年で特に映像産業において重要な技術となり始めている。しかしながら大域照明は従来のモデルに比べて計算負荷が高く、画像の生成に非常に時間がかかることが問題となっている。そのため、その重要性にも関わらず特に一般ユーザーにとって扱いにくいものとなってしまっている。

近年のコンピューターグラフィックスの発展において特筆すべき点として、大域照明を扱うようなオフラインレンダリングに対するものとして、リアルタイムレンダリングの処理速度や汎用性の加速度的な向上が挙げられる。特に、個人向け PC に搭載されるビデオカードではプログラマーによって処理内容をプログラムすることが可能な物が登場し、様々な処理をビデオカード上で行う事が可能となってきている。これらはプログラマブルシェーダーと呼ばれ、シェーディング処理（陰影処理）だけにとどまらず、その汎用性により数値計算などにも用いることが出来る。

本プロジェクトではこれらビデオカード（CPU に対応して GPU(Graphic Processing Unit)と呼ばれる事が多い）の高速性や、プログラマブルシェーダーの汎用性に注目し、大域照明をハードウェアによる支援を用いて計算することを考えた。

２．目的

大域照明は一般に CPU によるレイトレーシングで計算されることが多いため、まず GPU に適した大域照明の計算方法を開発する事を考えた。また、その手法を用いて、DirectX 9.0 の Vertex / Pixel Shader 2.0 をプログラマブルシェーダーとして使用した Windows 上で動作するスタンドアロンのレンダラー（レンダリング処理のみを担当するソフトウェア）を作成することを目的とした。それに加えてこのソフトウェアを Web 上で公開することで、GPU の大域照明計算における有用性を一般に広く知ってもらいたいという狙いもある。

３．従来の手法

大域照明計算は具体的にはレンダリング方程式[1]を解くことであるが、実際は解析的に解くことは出来ず数値計算で解くしかない。そのため一般的には Monte Carlo 法を用いた数値計算手法が用いられる。特に CG の分野では Ray tracing と呼ばれる、光の反射や屈折を幾何学計算により追跡するアルゴリズムによる数値計算を行ってレンダリング方程式を解くことが多い。Ray tracing は現在のところ専用のハードウェアによる処理はほとんど行われておらず、汎用的な CPU によって処理される。従って元々のアルゴリズム上の計算負荷に加えて、汎用的な CPU でしか処理を行えない事から、Ray tracing は非常に計算時間のかかるアルゴリズムとなっている。Ray tracing は計算負荷が高い処理ではあるが、各フラグメントの計算が独立しているため、処理の並列化による高速化が容易であり、クラスタリングによる高速なレンダラーが既に開発されている[6]。ところが、これらクラスタリングによるレンダリングは一般のユーザーにとってはコストの面で負担がかかるため、企業による映像制作などの場面でしか用いられていないのが実状である。

Ray tracing によりレンダリング方程式を解く方法は、レンダリング方程式の特性上非常に効率が悪い。従って計算物理学の分野での Metropolis 法を応用し適応型のサンプリングを行う Metropolis Light Transport[2]や、光源からの Ray tracing と視点からの Ray tracing の結果を効率よく統合することの出来る Photon Mapping[3]、光の経路によりサンプリング方法を変える Bidirectional Ray tracing[7]など、より効率よくレンダリング方程式を解く手法が研究されている。

Photon Mapping は特に最近注目されている手法で、光の経路により計算方法を変えることで、従来の Ray tracing のみによる方法よりも高速にノイズの少ない画像を得る事が出来る。Photon Mapping で光源からのトレースの結果として保存される Photon Map の生成は通常数秒程度であり非常に高速であるが、視点から実際にレンダリングする際に用いる Ray tracing が主なボトルネックとなっており、特に後述する間接照明の計算が処理時間のほとんどを占めている。本プロジェクトではこの点に着目し、Photon Mapping を用いたレンダリングのボトルネックを GPU により解消するような手法を考えた。

４．プログラマブルシェーダー

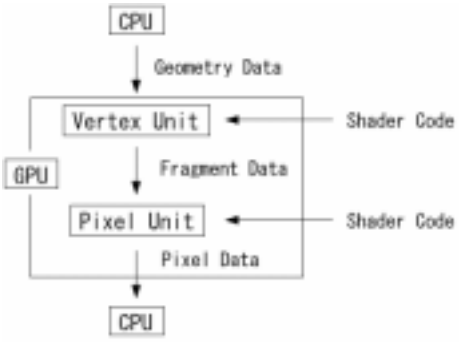

Fig.1 Programmable Shader

プログラマブルシェーダーとは CG の陰影処理の汎用性を目指して提案された規格の総称であり、2001 年以降からハードウェアでそれらを実現した GPU が登場した。近年では制限つき構造化アセンブラ（動的分岐、ループ、サブルーチン無し）による実装や、高級言語による実装[8][9]が可能となっている。概念図は Fig.1 のようになっており、一般的に頂点データを処理する Vertex Shader と、ラスタライジングによって生成されたフラグメントを処理する Pixel Shader に分けられる。プログラマブルシェーダーは本来、陰影処理（シェーディング）を行うことを目的に開発された物であるが、実際には陰影処理のみならず、さまざまな処理をすることが可能である。GPU によるプログラマブルシェーダーは SIMD（Single Instruction Multiple Data）命令で構成されており、頂点データやフラグメントごとに処理の並列化がなされているため、高速なベクトル演算器として動作する[10]。これらの GPU は一般のユーザーが十分に利用できる程に安価であり、統一した規格が定まっているために動作のための環境も整っている。それに加え、GPU の処理速度は CPU よりも遥かに速いスピードで進化しており、行列演算やベクトル演算での速度は既に CPU を遥かに上回っている。従って今後、大域照明計算のみならず、さまざまな数値計算への応用が期待できるものであると言える。本プロジェクトでは、GPU を利用するためのフレームワークである DirectX 9.0 によってシェーダーを実装する方法を選択した。

５．アルゴリズム

（1）従来の手法

GPU は大域照明計算に一般に使われる Ray tracing とは全く違う、z-buffer によるレンダリングを基礎としている。z-buffer によるレンダリングでは基本的にある面に射影したデータを用いて処理を行う。従ってフラグメント間に高い相関性があるため（現在注目しているフラグメントが、あるポリゴン内にある場合、隣のフラグメントも同じポリゴン内にある場合が多い）、ハードウェア上ではそれを利用して高速なレンダリングを実現している。一方、Ray tracing では各フラグメントの計算が完全に独立しているため、基本的な実装では相関性は全く利用されない。しかし計算が独立しているという事から、フラグメントごとに処理内容を変えることが出来るという利点がある。本プロジェクトでは、これらアーキテクチャの違いを考慮した上で新たな手法を開発した。

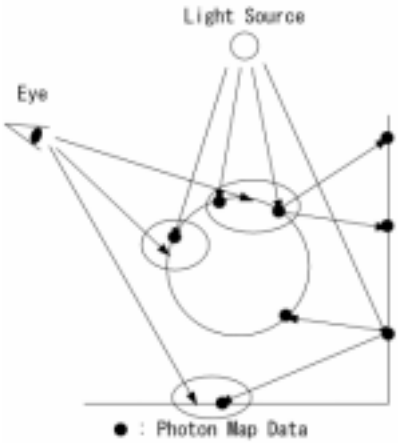

Fig.2 Photon Maping

まず、本手法の基礎となるのはフォトンマッピングについて概要を説明する。フォトンマッピングは光源からフォトンと呼ばれる光のエネルギーをトレースし、物体と光の相互作用が起こる点をフォトンマップとして保存しておき、視点からのレンダリング時に考慮する大域照明の計算手法である（Fig. 2）。これにより、視点からのレンダリング時に毎回、光源からのエネルギーを計算する必要がないために、単純なレイトレーシングによる計算より高速なレンダリングが可能となる。フォトンマッピングはフォトンの経路によってグローバルフォトンマップとコースティクスフォトンマップに分けることでより効率の良い計算が出来る[11]。これは、単純に全ての経路でのフォトンを格納したフォトンマップのみを扱うと、フォトンの密度が少ない点でアンダーサンプリングとなり、正確な結果が得られないという事実に基づいている。グローバルフォトンマップは全ての経路でフォトンを格納しており、コースティクスフォトンマップは集光現象や鏡による反射などの経路を持つフォトンのみを格納している。

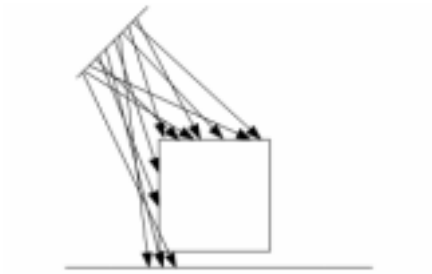

Fig. 3 Direct Illumination Rays

これら分割されたフォトンマッピングを用いてレンダリングを行う際には、直接照明、間接照明、コースティクスに分けた計算が行われる。まず、直接照明は光源から（実際には光源に向かって）レイを飛ばし、間に遮る物体が無ければ光が当たっているとして、それを光源全体で繰り返すことによって計算が行われる（Fig. 3）。これはノイズの無い結果を得るために多くのレイを飛ばす必要があるため、特に光源が大きい場合について時間がかかる処理である。

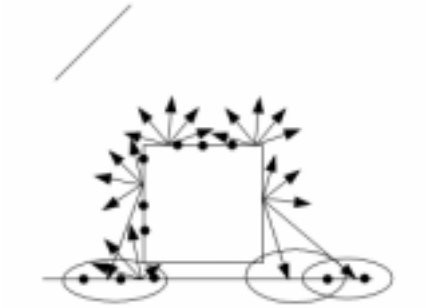

Fig. 4 Indirect Illumination Rays

次に間接照明は、各点に届く光をグローバルフォトンマップによって計算することで計算できる。具体的には各点に届く光がN回反射した物であるとすると、グローバルフォトンマップはN－1回反射した光のデータであると考えて、N－1回からN回までの光の経路をレイトレーシングすることで計算できる。この様子を図にするとFig. 4のようになり、各点でのレイの方向は半球状に分布している。これはノイズの無い結果を得るためには通常数百～数千のレイがフラグメント毎に必要となり、Photon Mappingによる大域照明計算の主なボトルネックとなっている。

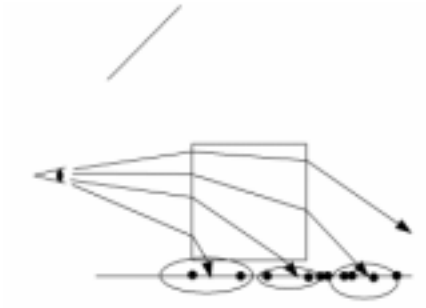

Fig. 5 Caustics Rays

最後に、コースティクスはコースティクスフォトンマップを参照し、各点でのフォトンの密度を求めることによって計算される（Fig. 5）。コースティクスフォトンマップは光のエネルギー分布が局所的であり、フォトンによって密度を求めても十分に正確な結果が得られる。この処理にはサンプリングする点での近傍にあるフォトンを求める必要があるため、木構造などによるフォトンデータのクエリーが必要となる。従って、本プロジェクトでは基本的なkd-treeによる空間分割[13]を行って高速なクエリーを実装している。

(2) 提案手法

以上のフォトンマッピングの手法に基づき、各計算でのレイの相関性に注目してGPUによるz-bufferレンダリングを利用できるような手法を開発した。以降から実際に本プロジェクトで実装した方法について説明する。

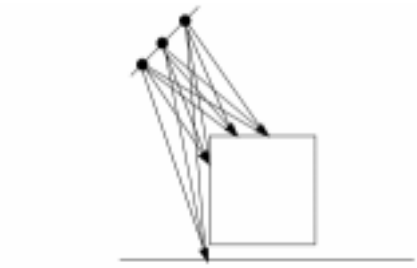

Fig. 6 Correlation of Direct Illumination Rays

まず、直接照明の計算は光源を点でサンプリングするとFig. 6のような光の方向の相関性がある事がわかる。この相関性を持った光の束は、z-bufferによる透視投影に対応させることが出来るため、これを用いて光のエネルギーが各点にどれだけ届いているかを計算することが出来る。大まかな処理手順は以下の通りである。

1. 光源から見える最も近い点までの距離をレンダリングして保存しておく。
2. 視点からの見える点の光源までの距離を計算する。
3. 1で保存した距離よりも2の方が遠ければその点は影になっているとし、それ以外は光が当たっているとする。

この手法はShadow Mappingと呼ばれる手法であり、GPU上で高速に行う事が出来る[12]。これを光源全体ついてサンプリングを何度も行うことで形のある光源も扱うことが出来る。サンプリングノイズは影の形に沿った帯状のノイズとなるので、これは従来のレイトレーシングによる手法での点ごとのノイズよりも目立たない物となっている。Shadow Mappingでは一般に手順1で生成した距離バッファのエイリアシングが問題となるが、本手法では何回も光源を点でサンプリングするため、スーパーサンプリングによるアンチエイリアシングが可

能である。

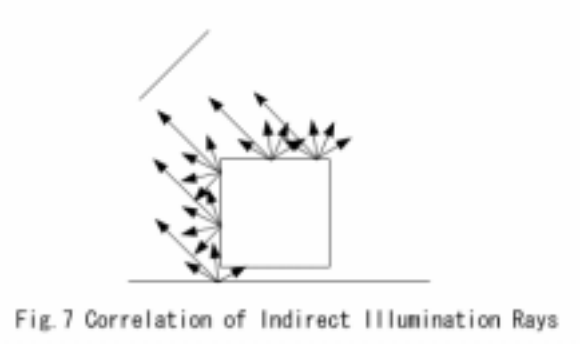

Fig.7 Correlation of Indirect Illumination Rays

次に、間接照明の計算に用いるレイに注目すると、各点である方向のみのレイを考えることで、原点の位置が違う平行なレイの束が考えられる（Fig. 7）。これはz-bufferで平行投影をすることに対応させることが出来るため、この方法により複数の点を一度にサンプリングする事が可能となる。実際には、各点の原点が異なるため単純な平行投影ではなく、各点から出た光が当たる最も近い点を求める必要がある。従って、Depth Peeling[14]と呼ばれる手法に基づき、以下のような方法を開発した。

1. ある方向にシーン全体を平行投影する。
2. その方向に沿って最も遠い面を描画して保存する。
3. 視点から見える点を平行投影した点と2で保存されている点の距離を比較し、2方が遠い場合に描画する。
4. 2のデータより近い面で最も遠い面を描画する。
5. 4のデータを2にコピーして3に戻り必要な回数繰り返す。

これを各方向について行うことである方向を見たときに各フラグメントに最も近い点をサンプリングすることが可能となる。従来のレイトレーシングによる手法では各点の各方向に個別にレイトレーシングを行わなければなかったため、この間接照明の計算がレンダリング時間の大きな部分を占めていた。この手法によれば、レンダリング対象となるフラグメントを一度にサンプリングしているため、GPU上でフラグメント毎に並列処理が行われる事になる。この際、グローバルフォトンマップを木構造によって毎回クエリーする処理は負荷が高すぎるため、あらかじめモデルデータ上でサンプリングしておき、レンダリング時にはその結果を参照する事でさらなる高速化を行った。

最後にコースティクスの計算は、コースティクスフォトンマップを利用するために木構造などによるフォトンデータのクエリーが必要となる。この場合、再帰や動的条件分岐をサポートしない現在のGPUで計算を行うよりは、CPUによって実装を行ったほうが効率良く計算できると考え、従来の手法と同じく実装はCPUで行った。CPUとGPUは並列動作することが可能であるため、GPU で直接照明と間接照明を計算すると同時に、CPUでコースティクスを計算することが可能である。

一般的に画像の解像度はモデルデータに対して十分でないために、エイリアシングが発生する。従って、本手法では各照明計算でピクセル内のサンプリング位置をずらしながら計算することにより、アンチエイリアシング処理を行った。この処理は画像全体に行わず、エイリアシングが発生するピクセルを事前に計算することで高速な処理を行っている。これは従来のGPUによるレンダリングのみでは行えないため（各ピクセルに相関性が無くなる）、エイリアシングが発生するピクセルのみにレイトレーシングを行うことで実装した。

６．結果

実際に本プロジェクトで提案した手法を用いてソフトウェアを作成した結果をFig. 8に示す。実装はDelphi 6.0で行い、プログラマブルシェーダーとしてMicrosoft DirectX 9.0のVertex / Pixel Shader 2.0を用いた。CPUはPentium 4 2.7GHz、GPUはATIのRadeon 9700 Proである。モデルデータの複雑さにもよるが、CPUによるRay tracingとMonte Carlo法を使った手法に比べて、10倍以上の高速化がなされている事がわかる（Fig. 9）。しかし、レンダリング時間はパラメータの調整やアルゴリズムの違いにより各レンダラーによって違うため、一概に高速化が達成されたと言うことは難しい。しかしながら、前述の通りGPUの処理速度は加速度的に増加しており、GPU上で大域照明計算を実装した事は、将来的な処理速度の大きな向上が望めると言うことでもあるため、十分な高速化が成されたと言える。画質の面でも従来のRay tracingによるレンダリングで大きな問題となっていた、ピクセルごとのサンプリングノイズ（画像全体ではホワイトノイズ状）が全く無くなっている。Fig. 10はFig.9のレンダリング結果を拡大したものであるが、提案手法の方ではほぼノイズの無い結果が得られている事がわかる。これは本手法が相関性を利用して大量のピクセルを一度にサンプリングするため、ノイズはより人の目に気づきにくい相関性を持った帯状のノイズとして現れるからである。また、本手法ではピクセルごとのアンチエイリアシングのサンプリング数が従来のRay tracingに比べて圧倒的に多いため、アンチエイリアシングについても遥かに良い結果が得られている。

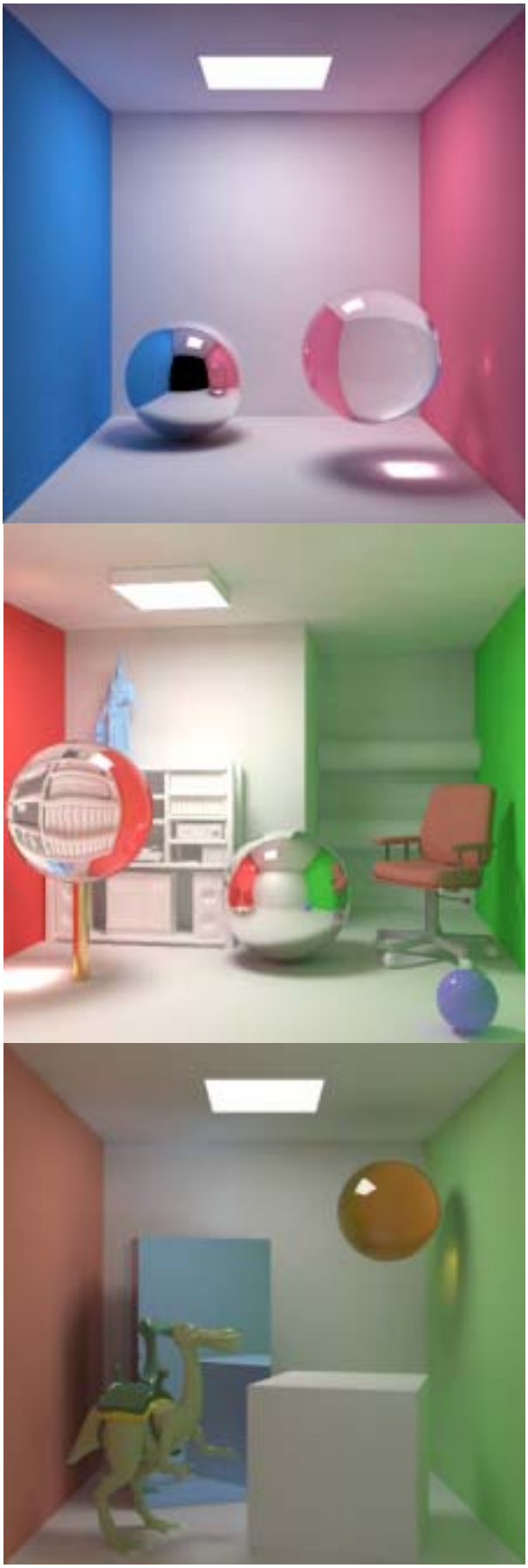

Fig. 8 Rendering Results of Test Scenes

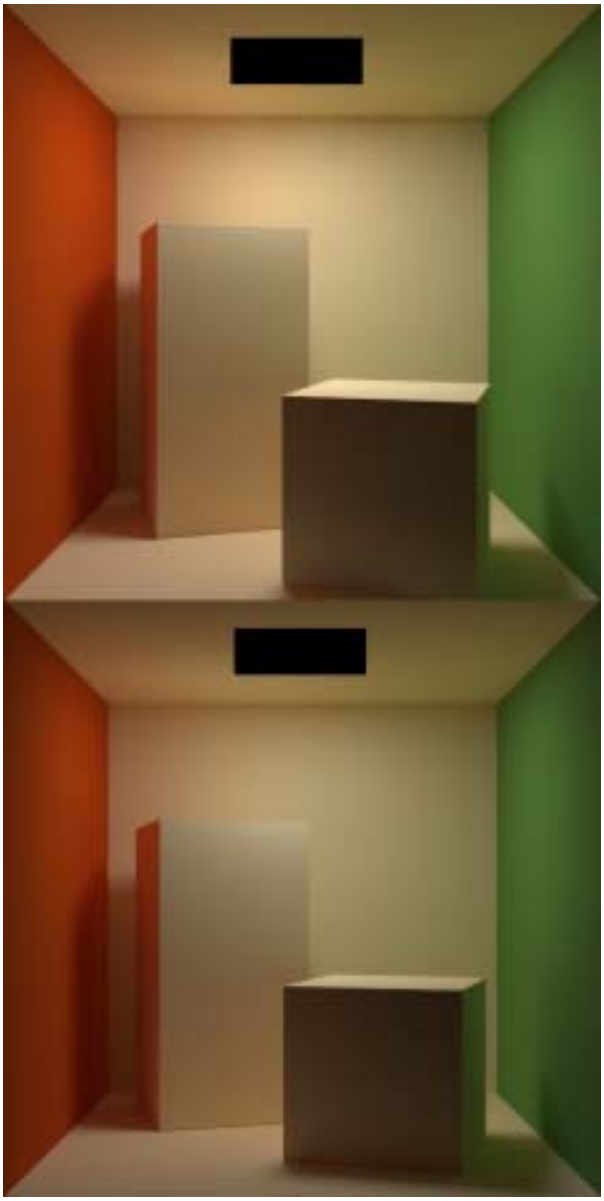

Fig. 9 Comparison of Rendering Results
Top: Proposed Method (Rendering Time: 238 sec.)
Bottom: CPU Raytracing (Rendering Time: 3543 sec.)

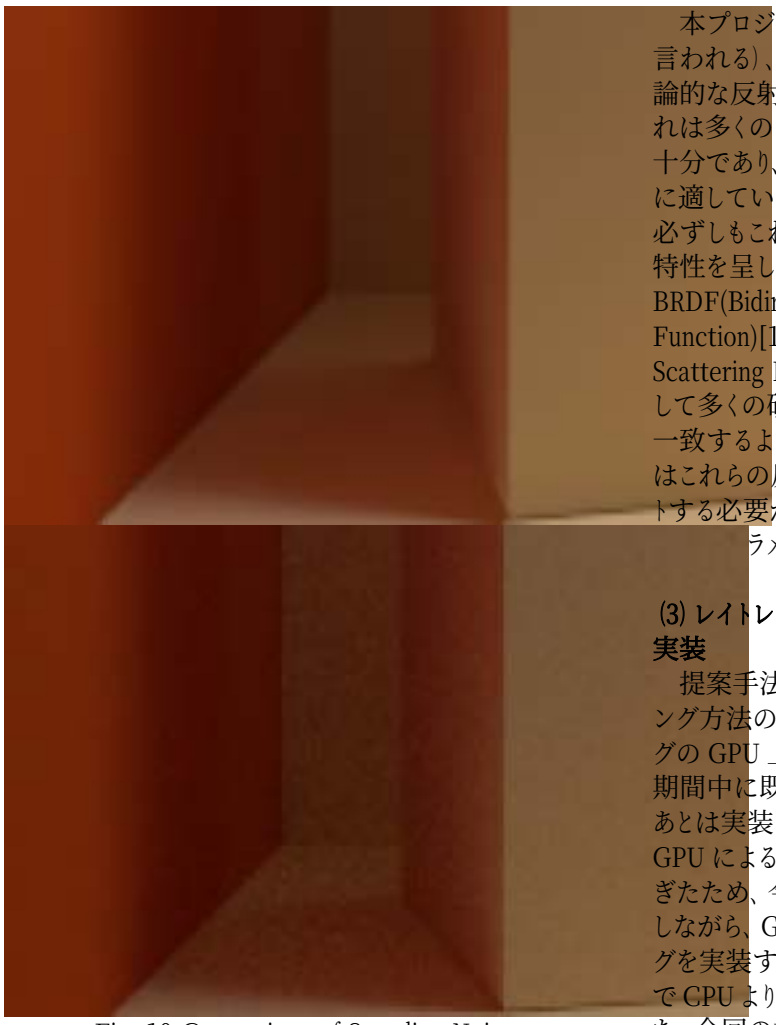

Fig. 10 Comparison of Sampling Noise
Top: Proposed Method
Bottom: CPU Raytracing

7. まとめと今後の発展

本プロジェクトで開発した手法により、従来の CPU のみによる手法に比べて十分な高速化が達成された。この手法は GPU に適した計算方法であり、GPU の処理速度の向上を考えると将来性のある手法であると言える。また、プログラマブルシェーダーを用いた大域照明計算の研究としての新たな可能性を提案できたと考えられる。また、今後の発展としては以下のような項目が挙げられる。

(1) 高速なプレビュー用のレンダリング手法の実装

本手法は従来のアルゴリズムより高速ではあるが、実際に使用する場合、さらに高速なプレビュー用のレンダリングが必要になる事が多い。従って、プレビュー用のより高速な手法を開発する必要がある。しかしながら、本手法は多少の変更により容易にインタラクティブなプレビュー用のレンダリングを行う事が出来るように設計されており、既に実装のための設計も完了しているため、今後は実際に実装してテストを行うのみとなっている。

(2) 複雑な反射特性のサポート

本プロジェクトでは完全拡散反射（ランバート反射と言われる）、及び完全鏡面反射・屈折という、いわば理論的な反射特性のみをサポートすることを考えた。これは多くのシーンでこの3つのみのサポートであっても十分であり、かつ今回サポートに含めたモデルデータに適しているからある。しかしながら、現実の物質では必ずしもこれらのモデルは正しくなく、より複雑な反射特性を呈している。これらの反射特性は CG の分野で BRDF(Bidirectional Reflectance Distribution Function)[15]（もしくは BSSRDF(Bidirectional Surface Scattering Reflectance Distribution Function) [16]）として多くの研究がなされており、既に物理的な実験に一致するようなモデルも開発されている。従って、今後はこれらの反射特性を実装しより複雑な反射もサポートする必要があると思われる。付け加えて、テクスチャパラメータの制御なども実装するべきである。

(3) レイトレーシング・フォトンマッピングの GPU 上での実装

提案手法以外で考えられる GPU を利用したレンダリング方法の一つに、レイトレーシング・フォトンマッピングの GPU 上での実装が挙げられる。本プロジェクトの期間中に既にアルゴリズムが理論上では完成しており、あとは実装する段階であったが、実際には現時点での GPU によるプログラマブルシェーダーの制限が大きすぎたため、今回はこの手法は用いない事にした。しかしながら、GPU 上でレイトレーシング・フォトンマッピングを実装することは、既にいくつかの研究[17][18]の中で CPU より高速になるであろう事が指摘されている。また、今回の提案手法では CPU によるレイトレーシング・フォトンマッピングの処理が GPU による処理よりも時間がかかっているため、この点が高速化されればさらに大幅にレンダリング時間を短縮することが出来る。従って、今後のプログラマブルシェーダーの発展に伴ってレイトレーシング・フォトンマッピングを GPU 上で実装するべきであると考えられる。

8．参加企業及び機関

株式会社　創夢（プロジェクト管理）

9．参考文献

[1] James T. Kajiya : The Rendering Equation. Computer Graphics, pages 143-150, August 1986.
[2] Erich Veach and Leonidas j. Guibas : Metropolis Light Transport. SIGGRAPH 97 Proceedings, pages 65-76, August 1997.
[3]Henrik Wann Jensen : Global Illumination using Photon Maps. Rendering Techniques ’96, pages 21-30, 1996.
[4]nVidia corporation : ‘Power of 3D’ > ‘Demos’. http://www.nvidia.com/view.asp?PAGE=power_demos
[5]Tomas Akenine-Moller and Eric Haines : Real-Time Rendering 2nd edition. A.K. Peters Ltd. ISBN 1568811829.
[6]Toshi Kato and Jun Saito : "Kilauea": parallel global illumination renderer. Pages 7-16. Eurographics

2002.
[7]Eric Veach and Leonidas J. Guibas : Optimally Combining Sampling Techniques for Monte Carlo Rendering. SIGGRAPH 95 Proceedings, pages 419-428. 1995.
[8]nVidia corporation : C for Grahphics. Cg shaders. http://www.cgshaders.org/
[9]Micorosoft corporation : DirectX 9.0 SDK Help. 'Direct X Graphic' > 'Reference' > 'Shader Reference' > 'High-Level Shader Language'.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/directx9cpp.asp
[10]Pat Hanrahan : Why is Graphics Hardware so Fast?
http://www.graphics.stanford.edu/~hanrahan/talks/why
[11]Henrik Wann Jensen, Frank Suykens, and Per Christensen:A Practical Guide to Global Illumination using Photon Mapping. SIGGRAPH 2001 Course 38, Los Angeles, August 2000
[12]nVidia corporation : Hardware Shadow Mapping. http://developer.nvidia.com/view.asp?IO=hwshadowmap_paper
[13]NIST(National Institute of Standard Technology) : kd-tree.
http://www.nist.gov/dads/HTML/kdtree.html
[14]Cass Everitt : Order-Independent Transparency. http://developer.nvidia.com/docs/IO/1262/ATT/OrderIndependentTransparency.pdf
[15]Szymon Rusinkiewicz : A Survey of BRDF Representation for Computer Graphics.
http://www.cs.princeton.edu/~smr/cs348c-97/surveypaper.html
[16] Henrik Wann Jensen, Steve Marschner, Marc Levoy, and Pat Hanrahan : A Practical Model for Subsurface Light Transport. Proceedings of SIGGRAPH'2001, pages 511-518, Los Angeles, August 2001
[17] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan : Ray Tracing on Programmable Graphics Hardware. ACM Transactions on Graphics. 21 (3), pp. 703-712, 2002. (Proceedings of ACM SIGGRAPH 2002)
[18] Vincent C. H. Ma and Michael D. McCool : Low latency photon mapping using block hashing. Proceedings of the conference on Graphics hardware 2002