2003－CG－112 (19)
2003／8／19

GPUを用いた粗い表面上での光源形状を反映したハイライト生成

木村仁尚[†] 床井浩平[‡]

フォトリアリスティックな画像を作成するために物体表面に生成されるハイライトの形状は，光源形状と物体表面の粗さを考慮して決定すべきである．しかし，光源形状や物体表面粗さを反映したハイライトの生成は計算コストが高いため，これまでリアルタイム画像生成への応用は困難であった．本研究では光源形状を反映した環境マップを微小面分布確率に基づいてランダムサンプリングする手法のプログラム可能なレンダリングハードウェアであるGPU上への実装を行った．これにより光源形状と表面粗さを反映した形状を持つハイライトをほぼリアルタイムに生成することが可能となった．

Highlight generation considering both of light source shape and the roughness on the surface using GPU

Masanao Kimura[†] Kohe Tokoi[‡]

The highlight that is generated on an object surface in order to create a photorealistic image should be determined in consideration of shape of light source and roughness of an object surface. However, since generation of the highlight based on the shape of a light source or the roughness of the object surface had high computing cost, it was difficult to apply them to the real-time rendering until now. We implemented the technique of random sampling of the environmental texture map that projected the shape of a light source by probability of microsurface distribution on the GPU that is the programmable rendering hardware. This made it possible to generate the highlight in which reflection of the shape of a light source faded by surface roughness in mostly real time.

1. はじめに

従来リアルタイム 3DCG においての光源には点光源や平行光源が用いられているが，それらによるハイライト生成には経験的なモデルを用いることが多い．現実においてハイライトは有限の大きさを持つ光源の映り込みであり，その形状は反射面の粗さに起因する広がりを持つ．

ハイライト生成において反射面が滑らかである場合は正反射による反射光のみが視点に届くが，反射面が粗い場合はさまざまな方向からの光が反射によって視点に届き，結果としてハイライトにぼけが生じる．これを正確に再現しようとした場合，環境マップの全ピクセルに対してサンプリング処理を行う必要がある．これは計算コストが非常に高く，リアルタイム処理には適さない．

しかし，近年のレンダリングハードウェアの性能向上は著しく，またプログラム可能なGPUにより様々なレンダリング手法や表現手法が可能となり，リアルタイムレンダリングにおけるリアリティを格段に向上させることが可能となった．

本稿では反射面の微小面分布に基づいて環境マップをランダムサンプリングすることにより，表面の粗さに起因するハイライト形状のぼけを再現する手法を提案する．

2. 処理手順

提案手法は光源をポリゴンモデルによって定義し，レンダリング対象となる物体の表面粗さはパラメータにより指定する．

ハイライト生成の際には光源の形状の映り込みを行うために，照射位置から見た天空にある光源をレンダリングし，レンダリング結果を環境マップとして用いる．そして物体表面の粗さを表現するために微小面分布関数と幾何減衰係数を加味して環境マップのサンプリングを数回行い，これによってハイライト成分を決定する．

† 和歌山大学システム工学研究科 / Graduate School of Systems Engineering, Wakayama University
‡ 和歌山大学システム工学部 / Faculty of Systems Engineering, Wakayama University

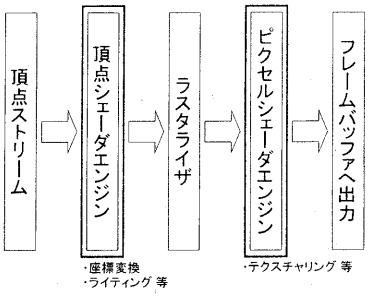


図1 グラフィックスパイプライン

環境マップのサンプリングを行う際には，バウンディングスフィアを用いて光源の存在しない領域のサンプリングを省く．一方，光源の存在する領域は表面粗さに基づいてランダムサンプリングを行う．

2.1 実行環境

NVIDIA 社製 GeForce FX 5200 を搭載した PC (CPU: Pentium4 2.6GHz, OS: Microsoft Windows XP Professional Service Pack 1) において DirectX 9.0a を用いた．頂点シェーダとピクセルシェーダは，ともにバージョン 2.0 を使用した．

2.2 GPU

GPU (Graphics Processing Unit) とはプログラム可能なレンダリング機能を持ったハードウェアである．近年のグラフィックスハードウェアの構成について簡単に説明する．グラフィックスパイプラインを図1に示す．入力された幾何データは頂点シェーダエンジンによって，座標変換やライティング計算などが行われる．その後ラスタライザによってピクセルデータに変換される．ピクセルシェーダエンジンはピクセルごとにデータを受け取り，テクスチャリングなどのピクセルごとの計算を行って最終的なピクセルの値を出力する．このうち頂点シェーダエンジンとピクセルシェーダエンジンがプログラム可能である．

GPU は複数のパイプラインが並列動作するため，ピクセルごとに異なる処理を行わせることや，ピクセルの描画順序に依存した処理は難しい．

GPU はアプリケーションソフトウェア側から GPU にプログラムを読み込ませることにより使用する．GPU にプログラムをロードした後に描画処理を行うことにより，GPU プログラムによるレンダリング結果が得られる．

GPU により実行可能なレンダリング手法には，トゥーンシェーダ[2]などの特殊な陰影付けモデルを用いたレンダリングを行うものや，レンズを通したような効果を得るものなどがある．高度な利用例としては波や炎などの自然現象のシミュレーション[3]や，レイトレーシングなどの本来プリレンダリングで用いられる手法をリアルタイムで実現させる試み[4]などがある．

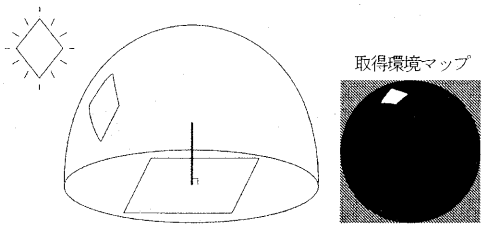


図2 環境マップによる光源形状の取得

一方，GPU をグラフィックスハードウェア上にある CPU のように捉え，レンダリング以外の処理を行わせる研究も行われている．しかし，本稿では GPU を本来の使用方法であるレンダリング機能の実装のために使用する．

3. 光源形状を反映したハイライト生成

ハイライトとは光源の形状が物体表面に映り込んだものであるため，現実的な形状を持つハイライトの生成を行うには光源の形状を無視することはできない．本稿の提案手法では光源に形状を持たせるために光源をポリゴンモデルで定義する．そして環境マップによりハイライトを生成するための映り込む光源形状をテクスチャとして取得する．

4. 粗い表面によるぼけの処理

ハイライトの広がりを正確に再現するためには，ハイライトが生成される物体表面の粗さを無視することはできない．提案手法ではこの物体表面の粗さを微小面分布確率によって表現する．微小面分布確率とは物体表面において基準面に対してある角度 δ をなす微小面が存在する確率である．この分布に従う微小面による鏡面反射の集まりとして粗い表面への映り込みを再現する．また本稿では微小面の法線ベクトルに対して通常の法線ベクトルを基準法線ベクトルと呼ぶ．

4.1 微小面分布確率

粗い物体表面上における微小面の向きのばらつきを統計的に表したものが微小面分布確率である．本稿の提案手法ではこの微小面分布確率を楕円外周部までの距離によって表現する．

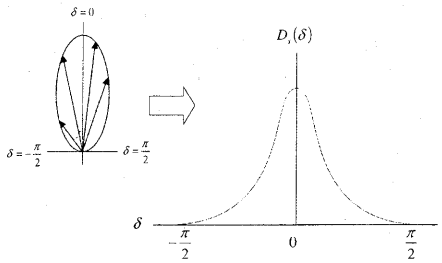


図3 楕円体分布

楕円による分布確率を用いる利点は δ の分布領域が[-π/2～π/2]であるため，微小面は基準法線ベクトルに対して角度[-π/2～π/2]の範囲のものが存在すると定義できること，そして楕円の長径と短径の比率 R_r によって物体表面の粗さを指定することが可能であることがあげられる．

楕円外周部までの距離を用いた微小面分布確率 D_s は次式で表現される．

$$D_s(\delta)=\frac{\sqrt{R_r^2-1}}{2R_r\tan^{-1}\left(\sqrt{R_r^2-1}\right)}\frac{\cos\delta}{\frac{\cos^2\delta}{R_l}+R_l\sin^2\delta}$$

$$\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}}D_s(\delta)d\delta=1$$

4.2 幾何減衰

粗い表面においては各微小面同士の遮蔽効果により鏡面反射光が必ず視点まで届くとは限らない．それは図のような現象が起きた場合である．

これらはシャドウイングとマスキングと呼ばれる．このマスキングシャドウイング効果による幾何減衰 G は次式で表される．

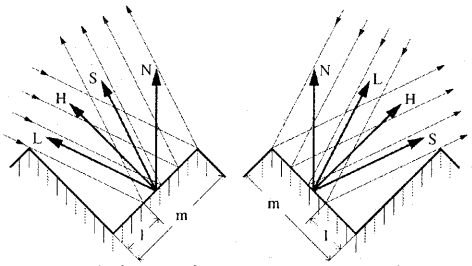
$$G=1-m/l$$

Torrance-Sparrow と Blinn はマスキングとシャドウイングの効果に関する G を決定している．視線方向ベクトルを S，光源方向ベクトルを L，微小面の法線ベクトルを H とし，それらが正規化されている場合，マスキングについての幾何減衰 G_m とシャドウイングについての幾何減衰 G_s は次式で表される．

$$G_m=\frac{2(N\bullet H)(N\bullet L)}{H\bullet L}$$

$$G_s=\frac{2(N\bullet H)(N\bullet L)}{H\bullet S}=\frac{2(N\bullet H)(N\bullet S)}{H\bullet L}$$

幾何減衰係数 G はこれらの値の最小値となる．



シャドウイング　　マスキング

図4 シャドウイングとマスキング

$$G=Min(1,G_m,G_s)$$

5. ランダムサンプリング

環境マップのランダムサンプリングによりハイライトの生成を高速に行う．サンプリングを行う微小面の法線ベクトルの算出は計算コストが高いため GPU で随時算出することはせず，あらかじめ微小面分布確率に従った微小面の法線ベクトルの分布をテクスチャとして作成しておき，乱数によりそのテクスチャをサンプリングすることにより微小面の法線ベクトルを取得する．

5.1 微小面分布確率法線マップ

微小面分布確率法線マップは楕円体分布乱数に基づいて微小面の法線ベクトルを取得するために使用する．テクスチャ座標 UV 座標系は極座標系に対応するように設定している．U 座標値[0.0～1.0]は一様乱数によって指定した値が微小面分布確率に沿った δ 値[-π/2～π/2]に，V 座標値[0.0～1.0]は一様乱数によって指定された値が一様分布の θ 値[0.0～π]に対応するものとしてテクスチャ要素の値を決定する．一様乱数の値である U 座標値から微小面分布に基づいた δ への変換は先に示した微小面分布確率関数を[-π/2～ δ]の範囲で積分した式 SD_s[-π/2, δ]を δ について解いた式へ U 座標値を代入することにより行える．

$$\delta(U)=\sin^{-1}\left(\frac{\tan\left((2U-1)\tan^{-1}\sqrt{R_r^2-1}\right)}{\sqrt{R_r^2-1}}\right)$$

$$U=[0.0\sim1.0],\quad \delta=\left[-\frac{\pi}{2}\sim\frac{\pi}{2}\right]$$

V 座標値から θ への変換はともに一様分布であるため，次式で得られる．

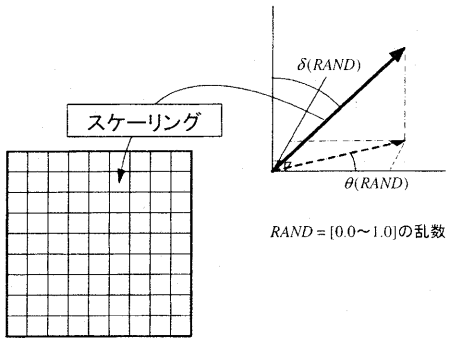


図5 微小面分布確率法線マップへの乱数の組み込み

$$\theta(V)=\pi V$$

$$V=[0.0\sim 1.0],\quad \theta=[0.0\sim \pi]$$

そしてテクスチャの値($R_{UV},G_{UV},B_{UV},A_{UV}$)には$\delta(U)$, $\theta(V)$の極座標値により求まる単位ベクトル(微小面の法線ベクトル)の直交座標系の値(H_x,H_y,H_z,1.0)を格納する．本稿では極座標の基準ベクトルはI_δ(1.0,0.0,0.0,1.0), I_θ(0.0,1.0,0.0,1.0)としている．またベクトル値の各値は[-1.0～1.0]でありテクスチャ値は[0.0～1.0]であるためそれを合わせるためのスケーリング変換も行う．

$$R_{UV}=\frac{H_x+1.0}{2.0}=\frac{\cos\delta(U)+1.0}{2.0}$$

$$G_{UV}=\frac{H_y+1.0}{2.0}=\frac{\sin\delta(U)\cos\theta(V)+1.0}{2.0}$$

$$B_{UV}=\frac{H_z+1.0}{2.0}=\frac{\sin\delta(U)\sin\theta(V)+1.0}{2.0}$$

$$A_{UV}=1.0$$

微小面の法線ベクトルをテクスチャの値に格納したことにより，法線ベクトルの各成分の分解能が[-1.0～1.0]/2^8 = 2.0/256まで低下する．通常のレンダリングであればこれによりエリアシングやモアレなどの問題が生じる可能性があるが，提案法ではランダムに法線ベクトルの決定を行っているため分解能の低下による影響はほとんどない．

5.2 ランダムサンプリング処理

実際にランダムサンプリングを行う段階において，乱数をもとに微小面分布確率法線マップを参照して微小面の法線ベクトルを取得するが，取得できる微小面の法線ベクトルは暫定的な基準法線ベクトルN_0(1.0,0.0,0.0,1.0)に対しての微小面分布となっているため，これを実際のレンダリング対象の物体表面の基準法線ベクトル $N(n_x,n_y,n_z,1.0)$に対しての微小面分布に変換しなければならない．この変換はN_0(1.0,0.0,0.0,1.0)から $N(n_x,n_y,n_z,1.0)$への回転であり，そのための変換行列はDirectXが左手座標系であることを踏まえて次のようになる．

$$\begin{bmatrix} n_x & n_y & n_z & 0.0 \\ 0.0 & n_z & -n_y & 0.0 \\ n_y^2+n_z^2 & -n_xn_y & -n_xn_z & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

また微小面の法線ベクトルはテクスチャの要素値として格納するために[0.0～1.0]の値に変換されているため，これをもとの値[-1.0～1.0]に戻す処理を先に行う必要がある．これを加味した変換行列は次のようになる．

$$\begin{bmatrix} 2.0n_x & 2.0n_y & 2.0n_z & 0.0 \\ 0.0 & 2.0n_z & -2.0n_y & 0.0 \\ 2.0(n_y^2+n_z^2) & -2.0n_xn_y & -2.0n_xn_z & 0.0 \\ -n_x-n_y^2-n_z^2 & n_xn_y-n_y-n_z & n_xn_z+n_y-n_z & 1.0 \end{bmatrix}$$

乱数によって微小面分布確率法線テクスチャを参照しこの変換行列を使用することによって，微小面の法線ベクトルを取得できる．そして微小面の法線ベクトルによる視線の正反射ベクトルを算出し環境マップのサンプリングを行う．以上の処理を1画素について数回行いハイライトの明るさを決定する．

5.3 GPU上での乱数発生

ランダムサンプリングを行うためにGPU上で乱数の発生が必要になる．乱数の発生には一般的に線形合同法が用いられているが，GPU上の変数はピクセル計算毎に初期化されるため，線形合同法を用いた場合同じ乱数が発生されてしまう．そのためGPUに適した乱数発生を行う必要がある．

提案手法ではGPU上で乱数を扱うために微小面分布確率法線テクスチャに乱数を組み込む．$\delta(RAND)$, $\theta(RAND)$としてテクスチャ値の算出を行い，座標値をもとに微小面分布確率法線テクスチャを参照することで表面粗さに基づいたランダムサンプリングを行っている．

6. バウンディングスフィア

ハイライトがほとんど発生しないポイントでの処理を省くために，そのポイントにおいてハイライト生成計算を行うべきか判定する処理を追加する．

実際の方法としては環境マップを作成する際に光源を光源の大きさよりある一定値大きい球で包む．その球はステンシルバッファにレンダリングをおこない．基準法線ベクトルによる視線の正反射ベクト

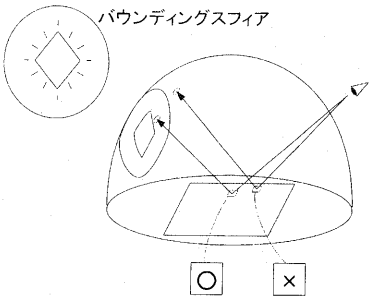


図6 バウンディングスフィア

ルがその球内を指し示した場合のみハイライト生成を行うように設定する．光源を包む球は環境マップ生成対象となるオブジェクト内に含まれる表面材質(表面粗さ)の数だけ用意する．

7. 実験結果

本稿で提案した実装手法の有効性を示すために，実行速度比較の実験を行った．表 1 は画像サイズ640×480，環境マップのサイズは256×256とした実行速度の比較である．

7.1 通常レンダリング

通常レンダリングとの実行速度比較のために，GPU，環境マップ，微小面分布確率を用いないハードウェアによるレンダリングを行った．

7.2 規則的サンプリング

ランダムサンプリングを用いない規則的なサンプリング点によるサンプリングをおこなった．基準法線ベクトルに対するサンプリング点の分布をあらかじめ決定しておき，サンプリングにより得られた輝度値は微小面分布確率 D_s に基づいて積算を行った．微小面分布確率 D_s もあらかじめ算出している．サンプリング点の数は GPU のプログラム長の制限により 13 点とした．

7.3 ランダムサンプリング

微小面分布確率に基づいてランダムサンプリングを行った．サンプリング点の数は 4 点とした．4 点としたのは 4×4 の行列計算を用いることによる高速化を考慮したためである．

8. おわりに

本稿では GPU を用いた光源形状と表面粗さを考慮したハイライト生成の実装手法を提案した．提案手法では物体表面の粗さを表す微小面分布確率を用いたランダムサンプリングにより，ハイライトのぼけを規則的なサンプリングよりも高速に再現することができた．またランダムサンプリングのための GPU での乱数の仕様についても提案を行った．

規則的サンプリング　ランダムサンプリング

図7 レンダリング結果の比較

通常レンダリング	約 530 fps
規則的サンプリング	約 28 fps
ランダムサンプリング	約 38 fps

表1 実行速度の比較

今後の課題として異方性を考慮することなどが挙げられる．

参考文献

[1] E. Lindholm, M.J. Kilgard, H. Moreton, "A User-Programmable Vertex Engine," Proc. SIGGRAPH 2001, 2001

[2] C.S. Marshall, "Non-Photorealistic Rendering (NPR): Real-Time 3D Cartoon Animation,", 2000 Game Developers Conference, 2000

[3] M.J. Harris, G. Coombe, T. Scheuermann, A. Lastra, "Physically-based visual simulation on graphics hardware,", Proc. SIGGRAPH 2002, 2002

[4] T.J. Purcell, I. Buck, W.R. Mark, P. Hanrahan, "Ray Tracing on Programmable Graphics Hardware," Proc. SIGGRAPH 2002, 2002

[5] Torrance, K.E., Sparrow, E.M., "Theory for Off-Specular Reflection from Roughened Surface," journal of the Optical Society of America Vol.57, 1967

[6] R. Ramamoorthi, P. Hnrahan, "An Efficient Representation for Irradiance Environment Map," Proc. SIGGRAPH 2001, 2001

[7] 佐藤いまり，佐藤洋一，池内克史，「全方位ステレオによる実光源環境の計測とそれに基づく仮想物体の実画像への重ねこみ」 電子情報通信学会論文誌 Vol.81 No.5

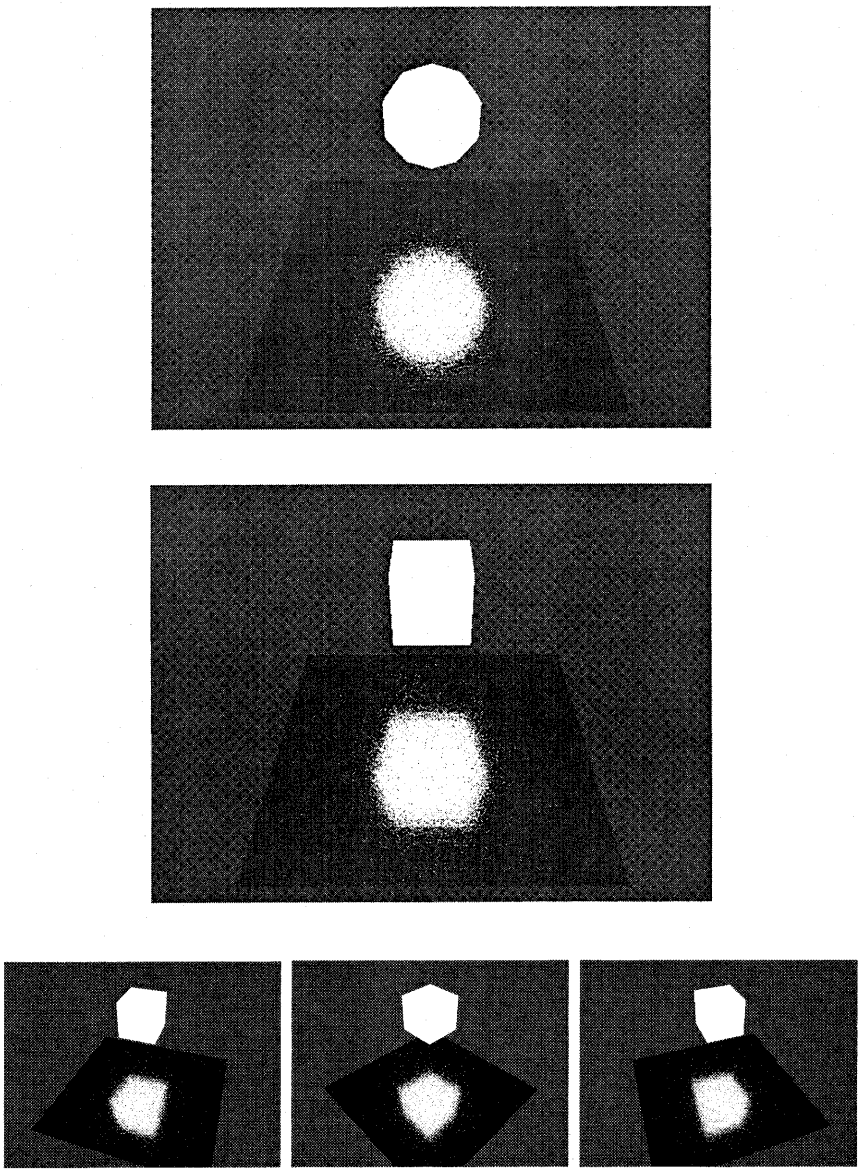


図8 提案法によるレンダリング結果