代理ジオメトリ宝石レンダリング

今給黎 隆

東京工芸大学 芸術学部 〒164-8678 東京都中野区本町 2-9-5
E-mail: t.imagire@game.t-kougei.ac.jp

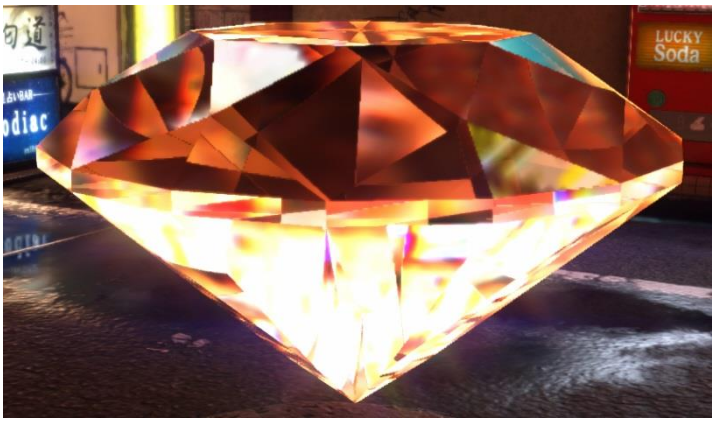

図 1　提案手法による宝石のレンダリング．© SEGA

概要　リアルタイムに描画可能な宝石のレンダリング手法を提案する．宝石を物理的にレンダリングする手法は以前から提案されているが，内部の複雑な形状による反射を処理する必要があるため，処理負荷が高く，ゲームなどのリアルタイムアプリケーションでは幅広く用いられていない．また，従来提案されていた手法も，追加のデータを必要としたり，マルチパスでのレンダリングが不可欠であるなど，追加のリソースを必要としていた．提案手法は，正確ではないが，追加のデータやパスを用意することなく，ソフトエッジで制作されたポリゴンモデルを用意してシェーダを差し替えるだけで，通常の物体と同様の描画手法で内部反射を考慮した宝石のレンダリングを実現する．提案手法は，各種ゲームエンジンに適応でき，宝石のリアルタイムレンダリングの品質を向上できるものと考えられる．

キーワード　宝石，レンダリング，リアルタイム

1.　はじめに

宝石のレンダリングはコンピュータグラフィックスにおいて新しいテーマではない．宝石の光学的な特性は古くから知られ，その理論的な原理も明確である．しかしながら，リアルタイムグラフィックスの世界において，宝石のレンダリングは普及しているものとはいえない．実際のビデオゲームでは，シャンデリアなどに関して，鏡面反射率を行う表面反射のみの不透明なマテリアルとして描画され，内部構造が反映されていない場合も多い（図 2）．実在の宝石では，光はその周波数に応じて異なる角度で屈折や反射を繰り返し，宝石内部を何度も行き来する．光線と宝石の面が交差する点は，レイキャスティングにより算出できるが，リアルタイムに計算するには負荷が高い処理である．交点の計算を省力化するための手法も提案されているが，追加のデータなしにリアルタイムに計算することは難しい．

我々は，ゲーム等のリアルタイムアプリケーションにおいて，追加のデータを必要とせず，高速に宝石をレンダリングする手法を提案する．宝石の内部構造を正確に計算しなくても結果は大きく変わらないことを仮定し，宝石の内部構造について描画するモデルの各ポリゴンから推測される代理のジオメトリで近似し，レンダリングを実行する．宝石内部の交点を代理のジオメトリの幾何学的な計算から導出することで，高速な交差判定を実現する．提案手法では，宝石のモデルをソフトエッジで作成し，専用のシェーダプログラムに差し替えるだけで，追加のデータ構造や描画パスを

図 2　従来のビデオゲームにおけるシャンデリアの表現の例．© SEGA

導入することなく宝石のレンダリングが実現できる．ゲーム制作の現場では，追加のデータ構造やパスの導入によりレンダリングプロセスを複雑化することは，新しい技術を導入する障害になることがあり，シンプルに導入できることは実際上重要である．

2.　関連研究

コンピュータグラフィックにおいて，宝石のレンダリングは古くから行われている．Sun ら[1]は，拡張されたレイトレーサーを用いて，ブリリアントカットのカラーダイアモンドのレンダリングを実現している．

Guy と Soler[2] は，宝石のレンダリングに関して，キューブマップなどのグラフィックスハードウェアの機能を利用することで，高速なレンダリングを実現している．しかしながら，木構造を使って内部反射で交差しうる面をカリングするものの，宝石を構築する多くのポリゴンとの交差判定をせねばならず，ゲーム等のリアルタイム性が強く求められるアプリケーションでは，計算負荷は低くはない．

Wyman[3]は，内部反射に関して，深度マップを利用して，交差判定を高速に処理する方法を提案した．また，Kalos[4]らも，屈折する物体に関してリフラクターマップを提案し高速に処理する手法を提案している．しかしながら，提案手法は，内部反射する物体の深度を画像の形式で事前にレンダリングする必要があり，レンダリングパイプラインに追加の工程を必要とする．レンダリング工程が増加して複雑化することは，全体的な最適化を妨げる可能性があるため，特に頻繁に使われることが少ない宝石のレンダリングにおいては望ましいことではない．

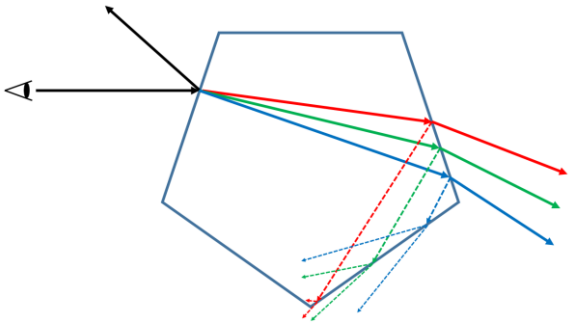

図 3　視点から飛ばした視線の屈折及び反射．

3.　宝石のレンダリング

宝石のレンダリングは光学的に説明することができる．宝石の表面に交差した光は，フレネルの式に従って，一部は反射し，それ以外は屈折して物質内部に入射する[5]．反射光の角度は入射光と等しく，屈折光はスネルの法則に従って，入射前と入射後の屈折率の比から求まる角度で進む[6]．宝石内部に侵入した光は，宝石の別の位置で再び表面と交差し，入射時と同様にフレネルの式に従った反射と屈折が再び生じる．宝石内部に入射した光は，最終的にはいずれかから表面で屈折して宝石外に射出されるが，光の一部は射出されるまでに宝石内の原子との相互作用により拡散され，宝石に色を付ける．処理の効率化として，一定の距離進むか，一定の回数反射した光の効果を無視する手法も提案されている[2]．ダイアモンドなどの宝石では，波長に応じて屈折率は異なり，入射する光の波長が短いほど，より強く曲げられる．これにより，宝石内部に入射した光は，射出時には周波数ごとに射出される場所と向きが変わり，複雑な虹色のグラデーションを産みだす[7]．

以上について，視点から各画素に視線を飛ばした場合を考える（図 3）．物体の表面で視線方向に入射する光は，入射点で反射してきた光と，透過してきた光の和となる．反射してきた光は，正反射方向から入射し，物体表面でフレネルの式で減衰したものである．透過してきた光は，光の周波数に応じた屈折率でスネルの法則により曲げられる方向から来た光であり，それら

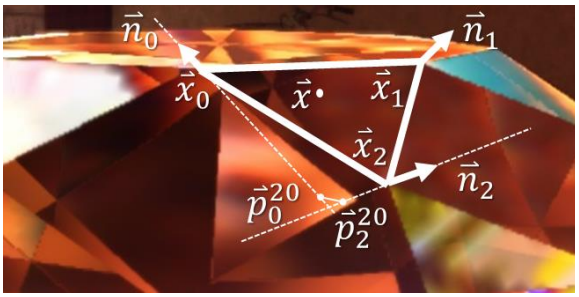

図4　宝石のモデルを構成するポリゴン及び2つの頂点に関しての法線方向の線分同士の最近接点.

も宝石の他の面において，反射してきた成分と，外部の光が屈折して入射してきた成分の足し合わせである.

4.　提案手法

提案手法では，前章での宝石のレンダリング手法に対して，宝石を三角形のポリゴンでモデル化し，視線が入射した位置におけるポリゴンを用いて宝石の内部形状を仮想的に構築して描画を行う.

4.1　代理ジオメトリの半径の推定

提案手法では，宝石内部の形状に関して，三角形が敷き詰められた形状（代理ジオメトリ）を仮定し，物体内部に侵入した後の反射面を近似する．最初に，宝石の大きさに関して，球の形状を仮定して推定を行う．視線が交差するポリゴンに関して，ポリゴンの i 番目 $(i = 0,1,2)$の頂点の位置座標と法線ベクトルを $\vec{x}_i$ と $\vec{n}_i$ とする(図 4). 法線ベクトルは頂点が関与する面の法線を平均化した共有法線とする．宝石が球であり，各法線が球の中央を向いていた場合，頂点から法線方向に伸びる線分は球の中心で交差する．しかしながら，一般の形状では線分の交差点は一致しない．それゆえ，線分の近接点の平均を宝石の大きさを推定する球の中心とする.

ポリゴンの各頂点を通り，法線方向を向く線分は，次の式で書くことができる

$$\vec{p}_i(t_i) = \vec{x}_i + \vec{n}_i t_i. \tag{1}$$

この線分間の最近接点は，線分間の距離

$$l^{ij}(t_i, t_j) = \left|\vec{p}_i(t_i) - \vec{p}_j(t_j)\right| = \sqrt{\left(\vec{p}_i(t_i) - \vec{p}_j(t_j),\ \vec{p}_i(t_i) - \vec{p}_j(t_j)\right)}, \tag{2}$$

を t_i, t_jで偏微分することで求められる. ここで $(\cdot,\cdot)$ は，ベクトルの内積である．偏微分の結果が 0 になるものとして最近接点を求めると，

$$\vec{p}_h^{ij} = \vec{x}_h + \frac{\left((\vec{x}_j - \vec{x}_i)\times\vec{n}_h, \vec{n}_i\times\vec{n}_j\right)}{\left(\vec{n}_i, \vec{n}_j\times(\vec{n}_i\times\vec{n}_j)\right)}\vec{n}_h, \tag{3}$$

が得られる (Appendix A)．ここで，hは， iもしくはjであり，$\times$ は 3 次元ベクトルの外積である．ポリゴンの面法線ではなく，共有法線を用いることにより，(3)式の分母が 0, すなわちポリゴンの各頂点で法線方向に通る線分が平行になることを避け，線分間の最近接点が導出される.

最近接点は線分毎に異なるため, 3 頂点の線分に関して 6 個の最近接点が存在する．計算の簡略化として，線分間の各最近接点 2 点に関して, 1 点のみを採用する．また，線分の向きが等しい場合に交点が無限遠方になるのを防ぐために，線分の向きで重みを付けて中心点を求める．最終的に，代理ジオメトリの中心 $\vec{o}$ は以下の 3 点の重み付き和で導出される.

$$\vec{o} = \frac{1}{W}\sum_{i=0}^{2} w_i \vec{p}_i^{ik}, \tag{4}$$

$$W = \sum_{i=0}^{2} w_i, \qquad w_i = |n_i \times n_k|. \tag{5}$$

ここで, kは, modを剰余計算として, $k = (i+1) \bmod 3$ で与えられるインデックスである.

4.2　反射面の決定

宝石内部で反射する面を構築する．宝石が交差した三角形の 1 つの頂点 $\vec{x}_0$に関して，代理ジオメトリの中心に対して対称な点 $\vec{x}_0' = 2\vec{o} - \vec{x}_0$ と，交差した面の面法線 $\vec{n}$ からなる平面 P を導入する(図 5). 視線の入射方向を$\vec{r}$とすると，視線は，宝石内部に侵入した際に，スネルの法則に従って，方向 $\vec{r}^t$ に屈折する[5],

$$\vec{r}^t = -\eta(\vec{n}\times(\vec{n}\times\vec{r})) - \vec{n}\sqrt{1 - \eta^2(1 - (\vec{n},\vec{r})^2)}. \tag{6}$$

ここで，η は，宝石へ入射する内外の媒質の相対屈折率であり，宝石の屈折率と，入射前の屈折率を n^{gem}, n^{out}として，下記の式で与えられる.

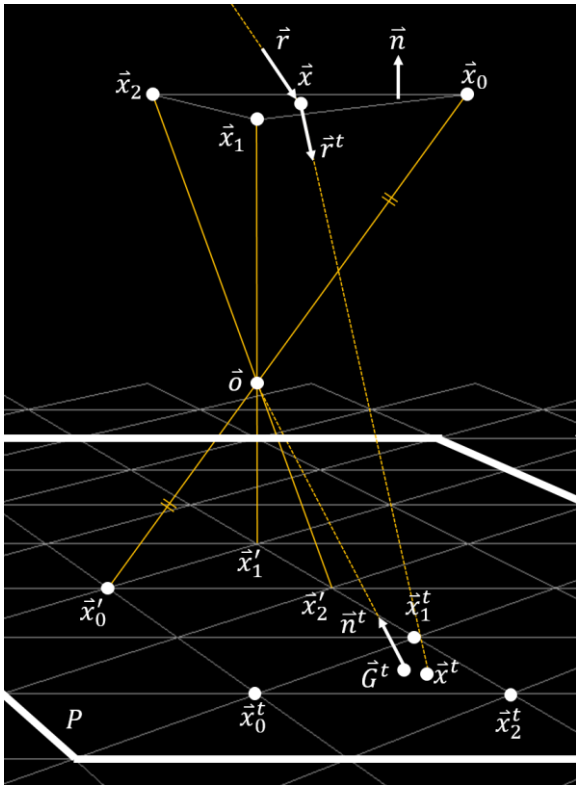

図 5　代理ジオメトリの交差面の構築.

$$\eta = \frac{n^{out}}{n^{gem}}. \tag{7}$$

屈折した後，視線の交差位置 $\vec{x}$ から $\vec{r}^t$ 方向に進むレイが平面 P と交差する点を $\vec{x}^t$ とする．平面 P 上で $\vec{x}'_0$ と，他の２頂点に関して中心 $\vec{o}$ を通って平面 P と交差する交点 $\vec{x}'_1$，$\vec{x}'_2$ のなす三角形を繰り返しパターンとして充填した際に，$\vec{x}^t$を含む三角形の頂点を $(\vec{x}^t_0 \quad \vec{x}^t_1 \quad \vec{x}^t_2)$ とする．この三角形の面の法線 $\vec{n}^t$ を，三角形の頂点座標の重心から代理ジオメトリの中心 $\vec{o}$ への方向により定義する，

$$\vec{n}^t = (\vec{o} - \vec{G}^t)/|\vec{o} - \vec{G}^t|, \tag{8}$$

$$\vec{G}^t = (\vec{x}^t_0 + \vec{x}^t_1 + \vec{x}^t_2)/3. \tag{9}$$

宝石内に屈折してきたレイは，位置 $\vec{x}^t$ で，向き $\vec{n}^t$ の面に交差したものと考えて，2 次反射及び屈折の処理を行う．

より高次の反射は，$(\vec{x}^t_0 \quad \vec{x}^t_1 \quad \vec{x}^t_2)$による三角形から新たな平面を作成することで取り扱うことができる．

面法線が等しく，隣り合う三角形に関して，それらの代理オブジェクトの中心 $\vec{o}$ が十分に近ければ各三角形が作る平面 P は等しくなり，また共有された稜線を持つことから，2 次反射の面法線が不連続に変化する平面 P 上の稜線が同じ位置に存在する事になる．これにより，隣り合う三角形は合成された一枚の平面のように連続的に内部構造をレンダリングする．

4.3　実装

提案手法では，ポリゴンから導出される代理ジオメトリを利用して計算を行う．DirectX 10 のシェーダモデル 4.0 以上の GPU で実装可能である．

4.3.1　描画プリミティブ

宝石の屈折・反射計算は，宝石のプリミティブをレンダリングする際に行う．宝石を描画する画素の位置を視線が交差する点とみなし，描画する際のポリゴンの各画素での深度値を用いて交差する点を 3 次元空間に復元して，視線の追跡を行う．

宝石のモデルは，ソフトエッジで制作することにより，各面法線のベクトルが合成された法線ベクトルとなり，共有法線のデータが準備される．

4.3.2　頂点シェーダ

頂点シェーダでは頂点座標や法線ベクトルをワールド空間に変換して，ジオメトリシェーダに出力する．

4.3.3　ジオメトリシェーダ

ジオメトリシェーダでは，ポリゴンの３つの頂点情報から(4)式によって代理プリミティブの中心 $\vec{o}$ を求めるとともに，ポリゴンの面法線を計算する．ピクセルシェーダへは，レンダリングに必要な，代理プリミティブの中心や面法線及び，全ての頂点座標と頂点法線を出力する．

4.3.4　ピクセルシェーダ

視点からレイキャストを行い，物体表面で屈折したレイについて，代理プリミティブとの交差判定を行う．最終的な入射光は，環境マップをサンプリングすることによりその強度が取得される．なお，今回の事例では，高速化のため，宝石内部での 2 次以降の反射は処

理していない．集められた入射光は，ゲームエンジンの自己発光成分として，G バッファ[8]に出力する．宝石表面で反射して視線に入射する光については，今回のシステムでは，ゲームエンジンのディファードレンダリングの処理により統一的に計算が行えたため，ポリゴンの面法線及び表面反射率を出力すれば，他の物体と同時に計算を行うことが可能であった．

4.4　分散

分散は宝石をきらびやかに見せる重要な要素である．周波数によって屈折する方向が変わるために，結果として虹色のグラデーションが宝石内に生じる．分散を生じさせるには周波数に応じて複数のレイを追跡すれば良いが，レイが交差するたびに反射と屈折の 2 種類のレイを追跡する必要があるため，処理負荷は高まる．

提案手法では，視線が宝石内に侵入したとき及び，反射・透過の時にどの方向に異なる周波数の光が進むのかを追跡することにより，計算負荷が低い分散効果を実現する．

視線が屈折した後の方向 $\vec{r}^t$ は，宝石に衝突する際の方向 $\vec{r}$ から(6)式で導かれる．宝石を透過する光は，$\vec{r}$ と $\vec{r}^t$ がなす面で分散する．この分散の方向を

$$\vec{d} = \frac{\vec{r}^t - \vec{r}}{|\vec{r}^t - \vec{r}|}, \tag{10}$$

により定義する．透過したレイがさらに宝石内部で反射，屈折する際は，$\vec{r}_0 = \vec{r}^t$ と $\vec{r}_\varepsilon = \vec{r}^t + \varepsilon\vec{d}$ (εは小さな数)のレイに関して，反射，屈折の計算を行う．その結果として得られる $\vec{r}_0'$，$\vec{r}_\varepsilon'$の変化量

$$\vec{d}' = \frac{\vec{r}_\varepsilon' - \vec{r}_0'}{|\vec{r}_\varepsilon' - \vec{r}_0'|}, \tag{11}$$

を，$\vec{x}^t$ における新たな分散の方向とすることで，分散する方向を追跡する．レイが宝石外に透過する際は，周波数 λ を追跡するレイに関して，周波数 λ_0 を基準の周波数として，λ_0に対する視線が射出する方向 $\vec{r}_0^{out}$，その時点での分散の方向 $\vec{d}^{out}$ と，宝石へ入射してからレイが射出するまでの距離をdとして，

$$\vec{r}_\lambda^{out} = \vec{r}_0^{out} + d\left(\frac{\lambda}{\lambda_0} - 1\right)\vec{d}^{out}, \tag{12}$$

を，最終的に周波数 λの光が入射した方向と近似して計算を行う．

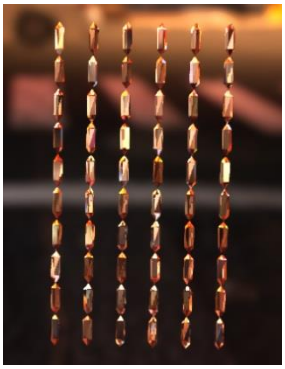

図 6　つなげられた五角柱の宝石．© SEGA

この近似による不整合は，異なる屈折率のレイが代理ジオメトリの三角形の異なるポリゴンに交差する際に発生する．三角形の境界をまたいだ際に，その後の反射・屈折方向は面の法線が異なるために大きく変化するが，提案手法ではその現象は生じない．したがって，代理ジオメトリの三角形の境界を跨ぐ画素に関して，本来よりも分散の効果が小さくなる．

なお，今回の結果の実装では，分光ベースのレンダリング[9]ではなく，λ_0を赤色の光の周波数として，それよりも大きな 2 つの周波数に関する色を緑と青の色のチャネルの強さに割り当てる簡易的な分光の計算を行っている．

5.　結果

本提案手法によるアプローチに関して，Intel Xeon CPU E5-1620 と NVIDIA GeForce GTX 760 のビデオカードを搭載した PC で検証を行った．使用したゲームエンジンは，株式会社セガゲームス内製のドラゴンエンジンであり，『龍が如く 6 命の詩。』(1)で使われている．レンダリングシステムには，タイルベースのディファードレンダリング[10]が採用されている．

図 1 はブリリアンカットのダイアモンドに対する提案手法の結果である．単なる表面の反射だけでなく，屈折光の内部構造に伴う反射により複雑な表情を見せていることが分かる．図 6 は，上下が閉じられた小さな五角柱をつなげた形状に関する結果である．角柱は

図 7　正確な内部形状による反射．© SEGA

縦方向にそれぞれの角度を変化させて配置しており，一回の描画命令の呼び出しで描画が可能である．

図 7 は, 図 1 と同じシェーディングアルゴリズムで，代理ジオメトリを用いずに，正確な内部形状による反射計算を行った結果である．実装としては，モデルの頂点情報をピクセルシェーダの定数として定義し，画素毎に全ポリゴンとの交差判定を行った．ブリリアントカットは，下部の形状が鋭くなっているため，特に宝石の上部における見た目が提案手法の結果よりも細かな造形となっている．

実行速度の比較として，1920×1080 の解像度において，全画面に宝石を表示した場合の速度を計測した．この場合において，GPU の処理に関しては，提案手法は 3ms で処理され，図 7 と同じ正確な内部形状による反射計算を行った場合は 15ms で処理が行われた．CPU 処理は共通であり，20 μ sec で処理された．また，シェーダの命令数に関する結果が表 1 である．共に十分な最適化を行っているとは言えないが，図 7 の手法と比較して，提案手法の方がピクセルシェーダの命令数は多いが，繰り返し命令を使用するピクセルシェーダ内でのポリゴンの探索を行わないため，画面に大きく表示してもシェーダ部分がボトルネックになることはない．また，正確な内部形状で計算する場合は，幾何学情報をピクセルシェーダの定数バッファに確保することになるため，必要となる定数バッファのリソースも多い．

表 1 提案手法と正確な内部形状による反射の方法(図 7)とのシェーダのコンパイル結果の差異.

	提案手法	正確な内部形状による反射
頂点シェーダ命令数	16 スロット	16 スロット
ジオメトリシェーダ命令数	160 スロット	160 スロット
ピクセルシェーダ命令数	181 スロット	152 スロット
ピクセルシェーダ定数サイズ	2304 バイト	15056 バイト

6.　まとめと今後の課題

軽量な宝石のレンダリング手法を提案した．提案手法は追加のリソースやレンダリングパスを必要とせず，アセットをソフトエッジで作成してシェーダを差し替えるだけで内部構造を持った屈折を行う宝石をレンダリングすることができる．

提案手法の課題としては，内部構造が三角形メッシュに制約されているという問題がある．代理ジオメトリの中心近くを通過して透過する場合には，隣接するポリゴンの法線が等しい場合に隣のポリゴンと反射・屈折する方向が連続するため，内部構造が多角形のように見える場合もあるが，原則的には内部構造は三角形ポリゴンの集合と観察されてしまう．また，提案手法は，内部構造がモデルの作り方に影響を受ける．鋭角なポリゴンを通して宝石内部に侵入したレイが交差すると，内部形状も鋭角な三角形となる．正確な内部構造の再現は今後の課題である．

7.　謝辞

本研究は，株式会社セガゲームスコンシューマ・オンラインカンパニー 第 1 CS スタジオのサポートにより進められ，論文の作成には，コニカミノルタ科学技術振興財団に助成をいただいた．

文　献

[1] Yinlong Sun, F. David Fracchia, and Mark S. Drew (2000). Rendering diamonds. In Proceedings of the 11th Western Computer Graphics Symposium (WCGS), 9–15.

[2] Stéphane Guy, and Cyril Soler (2004). Graphics Gems Revisited: Fast and Physically-based Rendering of Gemstones. *ACM Trans. Graph.*, 23(3), 231-238.

[3] Chris Wyman (2005). An Approximate Image-Space Approach for Interactive Refraction. *ACM Trans. Graph.*, 24(3), 1050-1053.

[4] László Szirmay-Kalos, Barnabás Aszódi, István Lazányi, and Mátyás Premecz (2005). Approximate Ray-Tracing on the GPU with Distance Impostors. Computer Graphics Forum 24(3) 695-704.

[5] Max Born, and Emil Wolf (1999). Principles of Optics, 7 ed. Cambridge University Press.

[6] Georg Beyerle, and I. Straut McDermid (1998) Ray tracing formulas for refraction and internal reflections in uniaxial crystals. Applied Optics 37, 34, 7947-7953.

[7] Günter Wyszecki, and Walter Stanley Stiles (1967) Color Science: Concepts and Methods, Quantitative Data and Formulas. Wiley.

[8] Takafumi Saito, and Tokiichiro Takahashi (1990) Comprehensible Rendering of 3-D Shapes. *ACM Trans. Graph.*, 24(4), 197-206.

[9] Robert L. Cook, and Kenneth E. Torrance (1982). A reflection model for computer graphics, ACM TOG 1, 7-24.

[10] Johan Andersson (2009). Parallel graphics in frostbite - current & future. SIGGRAPH Course: Beyond Programmable Shading.

ゲーム

(1) 『龍が如く 6 命の詩。』，株式会社セガゲームス，2016.（PS4）

Appendix A. 線分間の最近接点

(2)式について t_i，t_jで偏微分を行うことで，(3)式を導出する．(2)式の偏微分を行うと，次の式が得られる

$$\frac{\partial l^{ij}(t_i,t_j)}{\partial t_i}$$

$$= \frac{1}{2l^{ij}(t_i,t_j)}\frac{\partial}{\partial t_i}\left(-\vec{x}_{ji}+\vec{n}_i t_i-\vec{n}_j t_j, -\vec{x}_{ji}+\vec{n}_i t_i-\vec{n}_j t_j\right)$$

$$= \frac{1}{l^{ij}(t_i,t_j)}\left[-(\vec{x}_{ji},\vec{n}_i)+(\vec{n}_i,\vec{n}_i)t_i-(\vec{n}_j,\vec{n}_i)t_j\right], \quad \text{(A.1)}$$

$$\frac{\partial l^{ij}(t_i,t_j)}{\partial t_j} = \frac{1}{l^{ij}(t_i,t_j)}\left[(\vec{x}_{ji},\vec{n}_j)+(\vec{n}_j,\vec{n}_j)t_j-(\vec{n}_j,\vec{n}_i)t_i\right]. \quad \text{(A.2)}$$

ここで，$\vec{x}_{ji}=\vec{x}_j-\vec{x}_i$ を導入した．

最近接点は，(A.1)式，(A.2)式が共に 0 となる場合であり，[]の中の値を0とすると，

$$\begin{bmatrix}(\vec{n}_i,\vec{n}_i) & -(\vec{n}_j,\vec{n}_i)\\ -(\vec{n}_j,\vec{n}_i) & (\vec{n}_j,\vec{n}_j)\end{bmatrix}\begin{bmatrix}t_i\\ t_j\end{bmatrix}=\begin{bmatrix}+(\vec{x}_{ji},\vec{n}_i)\\ -(\vec{x}_{ji},\vec{n}_j)\end{bmatrix}, \quad \text{(A.3)}$$

と，まとめることができる．これを解くと

$$\begin{bmatrix}t_i\\ t_j\end{bmatrix}=\frac{1}{\Delta}\begin{bmatrix}(\vec{n}_j,\vec{n}_j) & (\vec{n}_j,\vec{n}_i)\\ (\vec{n}_j,\vec{n}_i) & (\vec{n}_i,\vec{n}_i)\end{bmatrix}\begin{bmatrix}+(\vec{x}_{ji},\vec{n}_i)\\ -(\vec{x}_{ji},\vec{n}_j)\end{bmatrix}$$

$$=\frac{1}{\Delta}\begin{bmatrix}(\vec{n}_j,\vec{n}_j)(\vec{x}_{ji},\vec{n}_i)-(\vec{n}_j,\vec{n}_i)(\vec{x}_{ji},\vec{n}_j)\\ (\vec{n}_j,\vec{n}_i)(\vec{x}_{ji},\vec{n}_i)-(\vec{n}_i,\vec{n}_i)(\vec{x}_{ji},\vec{n}_j)\end{bmatrix}, \quad \text{(A.4)}$$

となる．ここで，$\Delta=(\vec{n}_i,\vec{n}_i)(\vec{n}_j,\vec{n}_j)-(\vec{n}_j,\vec{n}_i)^2$を導入した．

外積の公式 $\vec{A}\times(\vec{B}\times\vec{C})=(\vec{C},\vec{A})\vec{B},-(\vec{B},\vec{A})\vec{C}$ 及び $(\vec{A}\times\vec{B},\vec{C}\times\vec{D})=(\vec{A},\vec{C})(\vec{B},\vec{D})-(\vec{A},\vec{D})(\vec{B},\vec{C})$ を用いると，次の結果が得られる．

$$\Delta=\vec{n}_i\cdot\left(\vec{n}_j\times(\vec{n}_i\times\vec{n}_j)\right), \quad \text{(A.5)}$$

$$\begin{bmatrix}t_i\\ t_j\end{bmatrix}=\frac{1}{\Delta}\begin{bmatrix}(\vec{x}_{ji}\times\vec{n}_j,\vec{n}_i\times\vec{n}_j)\\ (\vec{x}_{ji}\times\vec{n}_i,\vec{n}_i\times\vec{n}_j)\end{bmatrix}. \quad \text{(A.6)}$$

最終的に，(A.6)式に(A.5)式を代入して得られるt_i，t_jを(1)式に代入すると，(3)式が得られる．

Proxy Geometry Gem Rendering

Takashi IMAGIRE

Faculty of Art, Tokyo Polytechnic University 2-9-5 honcho, Nakano-ku, Tokyo, 164-8678 Japan

E-mail: t.imagire@game.t-kougei.ac.jp

Abstract We propose a rendering method for Gems in real-time. Physically based rendering for gems is not new. Since the processing load is high due to the need to determine the intersection points scattered in multiple internally, these methods have not been used yet widely in games. Also the conventional methods have required additional resources likes additional data structures or multipath rendering. Our method is not accurate but realizes a rendering gem considering internal reflection only replacing the shader without additional data or path. It is necessary to perform modeling soft edges, it is

possible to draw like a ordinal material. The proposed method can adapt to various game engine, it is possible to improve the quality of the gems of the real-time rendering.

Keywords Gem, Rendering, Real-time