社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－CG－122（13）
2006／2／20

デプスマップを用いたレリーフマッピングの高速化

鈴　木　健　司†

本論文ではデプスマップを用いたレリーフマッピングの高速化手法を提案する。レリーフマッピングではポリゴン上に貼られた高さマップに対してレイキャスティングを行い、単純な平面に正確な凹凸を表現することが可能である。本提案では、前処理として高さマップを囲む半球上のサンプル点から、高さマップを見たときの深さ情報をデプスマップとして作成する。そして、デプスマップの情報を用いて近似解を求め、そこからレイキャスティングを行うことによって交差判定を減らすことを可能とした。また、前処理によって作られたデプスマップ群のデータを効率よく減らすことでメモリコストを抑えた。さらに本稿ではデプスマップ群を利用して凹凸の作る影や凹凸によるアンビエントオクルージョン効果を計算できることを示す。

An Accelerated Relief Mapping Using Depth Maps

Takeshi SUZUKI†

In this paper, we propose a method that uses depth maps to accelerate relief mapping. The relief mapping itself renders microasperity on a plane surface by ray-casting a height map which is mapped to the surface. Our method pre-computes depth maps of a height map measuring from sampling points on a hemi-sphere over the height map. The method reduces computational cost of ray-casting by originating ray-casting from an approximate point which is computed by using the depth maps. Data size of the depth maps is downscaled to achieve a low memory cost. In addition, we present techniques that use depth maps to render shadows and ambient occlusion effects caused by microasperity.

1. はじめに

コンピュータグラフィックスの長い歴史において、物体の表面の凹凸を複雑な形状を定義せずに表現することは重要な研究課題とされてきた。その研究課題の先駆けとして Blinn のバンプマッピング[1] や Cook のディスプレイスメントマッピング[2] がある。これらの技術では比較的単純な形状の表面に、テクスチャによって凹凸を与え、複雑な形状を定義する。また、凹凸の作る局所的な影を表現するためのホライズンマッピング[4] や、視差を考慮した凹凸を表現するためのパララックスマッピング[3] などの方法が提案されてきた。

近年の研究としては Policarpo らが提案したレリーフマッピング[6)5)] がある。この技術ではポリゴンに貼られた高さマップに対し、レイキャスティングを行うことで、バンプマッピングなどでは不可能な正確な凹凸と輪郭を表現できる。しかし、精度の高い凹凸を計算するために高さマップへのアクセスが増え、レンダリング速度が低下してしまうという欠点がある。そこで本稿では、高さマップへのアクセスを減らすことによって、レリーフマッピングを高速化する手法を提案する。提案法の特徴は以下の３点である。

- 高速なレリーフマッピング
- 高さマップ以外の使用メモリが少ない
- 影やアンビエントオクルージョン (AO) への対応

本提案では前処理として高さマップのデプスマップを作成し、それを用いて交点の近似解を求める。そして、近似解を利用し、レリーフマッピングのレイキャスティングに必要な計算を減らすことで高速化する。本稿ではその手法と、デプスマップ群のデータサイズを、近似解の精度をほとんど失わずに減らせることを示す。また、前処理に使われたデプスマップで影や AO を計算できることを示す。

2. 関連研究

バンプマッピング[1] では物体表面の法線ベクトルを傾け、陰影を変化させることで、平面に凹凸があるかの様に見せることができる。少ない計算量で凹凸を表現できるが、凹凸による自己遮蔽や輪郭の変化、凹凸

† 法政大学 大学院 情報科学研究科
Graduate School of Computer and Information Sciences, The University of Hosei

が作る影などを表現することができないのが問題である。ホライズンマッピング[4] ではバンプマッピングにおける影の問題を解決した。

ディスプレイスメントマッピング[2] はポリゴンを微細なポリゴンに分割した上で凹凸を加える方法で、自己遮蔽や輪郭、影などを自然と表現することが出来る。この手法では莫大な微細平面をレンダリングする必要があり、実時間レンダリングには向いていない。

Kaneko らは視差により生じる視覚像の変化を表現する方法としてパララクスマッピング[3] を提案した。この手法ではテクスチャ座標を高さマップに応じて視線方向にずらす近似的な計算をしている。そのため、大きい凹凸の変形を表現するには向いていないが、計算のコストをかけずに凹凸を描画することができる。

Wang らは視点から凹凸を加えられる点までの距離を前処理段階で計算し、それを用いて凹凸をレンダリングする手法を提案した[7]。距離のデータは複数の視点から計算され、5次元関数のデータ群としてメモリに格納される。実際に GPU において実装する場合、莫大なメモリデータを圧縮する必要がある。この手法では実時間で精度の高い凹凸表現が可能であるが、使用するメモリ量が多くなるという問題がある。

2.1 レリーフマッピング

本節では本提案の基礎となるレリーフマッピング[6] について説明する。この手法ではポリゴンに対して彫る方向での凹凸表現が可能である。また、バンプマップなどではできない凹凸による自己遮蔽や影、正しい凹凸の輪郭などが表現できる。この手法ではまず初めにポリゴン上に貼られた高さマップに対して、描画する点を貫く視線によってレイキャスティングを行い、最も近い交点を求める。そして、交点におけるテクスチャ座標の高さマップや法線マップを使用することで、ピクセル単位に凹凸を表現する。なお、これらの処理はテクスチャ空間と高さマップの高さによって作られるハイトフィールド(u, v, h)で行われる。図 1(左) に凹凸を形成するのに必要な高さマップ示す。高さマップのデータは$0.0 - 1.0$で正規化されている。次にレリーフマッピングの具体的な処理手順を示す。

(1) 視点から描画点までの視線ベクトルを求める。
(2) 視線ベクトルを描画点の接線空間に変換する。
(3) 描画点を深さ 0.0 のスタート基点Aとし、変換された視線ベクトルを使い、深さ 1.0 の底面との交点 B 点を計算する。
(4) A 点から B 点まで、ある一定の進度で視線ベクトルを進め、高さマップとの交点を計算する。
(5) より正確な交点をバイナリサーチで求める。

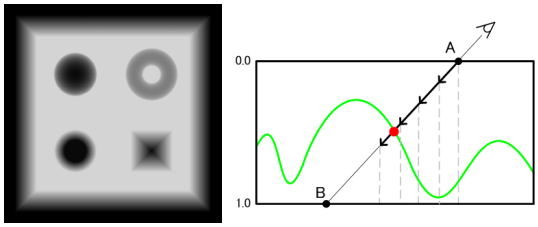

図 1 高さマップ (左) とハイトフィールドでの交差判定 (右)

(6) 交点におけるテクスチャ座標を求め、テクスチャや法線マップを使い、陰影計算を行う。

図 1(右) にテクスチャ座標を水平に見た時のハイトフィールドの様子と、交点を求める方法を示す。ハイトフィールド上では、視線ベクトルと凹凸との交差判定は高さの比較によって行われる。一定の進度で基点Aから視線ベクトルを進めて行き、凹凸の高さより低い位置に視線があった時、交差したことになる。なお、視線ベクトルの進度は深さ 1.0 を手動で設定された総ステップ数で割られた高さ分だけ進めている。視線が凹凸に潜り込んだ時、一つ手前の点との間でバイナリサーチを行うことで、より正確な交点を求められる。

レリーフマッピングではレイキャスティングを行うことで、少ないメモリ使用量で正確な凹凸を表現できるが、問題点もある。凹凸の細かさに比べて視線ベクトルの進度が長い場合や、視線が表面に対して平行に近い場合は視線ベクトルの進度が長くなってしまい、凹凸を飛び越え、誤った交点を見つけてしまう。

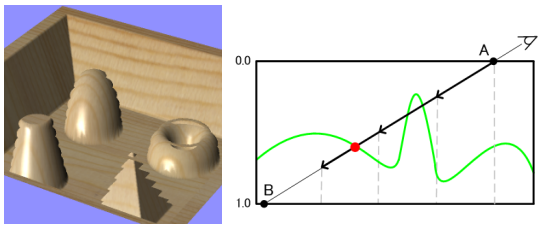

図 2 誤った交点計算による画像 (左) と交差判定の挙動 (右)

図 2 では誤った交点を見つけた時の挙動と、その結果の画像を示す。左の画像に示されるように三角錐の先や凹凸の輪郭が抜けてしまい、後ろの面が誤って表示されている。総ステップ数を増やし、視線ベクトルの進度を短くすることでこの問題を防ぐことができるが、交点を見つけるまでの交差判定の回数が増えてしまう。また、交差判定で高さマップにアクセスする時にバイリニア等のフィルタリングを使うと、メモリアクセスが大きなボトルネックになり、交差判定の回数の増加も問題になる。本提案では、より少ない交差判定の回数でレンダリングを行うことが出来る。

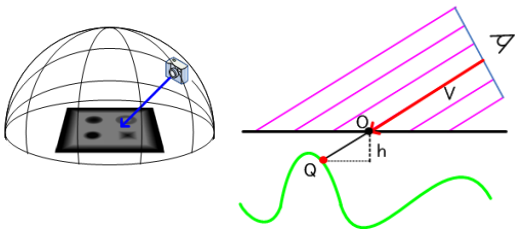

図 3 高さマップを囲う半球状のサンプリング点 (左) と前処理を行う時のハイトフィールドの様子 (右)

3. デプスマップを用いたレリーフマッピング

この節では本提案手法で行う前処理と、その結果を用いたレリーフマッピングの高速化手法を説明する。また、前処理で作成したデプスマップのデータサイズを、精度を保ちながら減らす手法を示す。

3.1 前 処 理

本提案では前処理として、サンプリング点から見た高さマップの値を高さマップが表現する凹凸の自己遮蔽を考慮しながらデプスマップに格納する。サンプリング点は図 3(左) のように高さマップを囲う半球を一定の角度 (極座標系でのθとϕ) で分割した点上に置く。図 3(右) は前処理を行う時のハイトフィールドの様子である。高さマップの表面の点をOとし、サンプリング点から点Oを見た時のベクトルをVとする。この時、点Oを通り、ベクトルVの方向を持つ半直線と最初に凹凸と交差する点をQとする。我々はこの点Qの基準面からの深さ (h) を前処理で計算し、デプスマップに格納する。

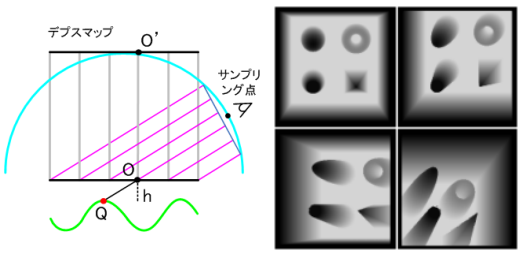

図 4 デプスマップに格納される深さ情報 (左) と前処理で生成されたデプスマップの例 (右)

図 4(左) に深さhをデプスマップに格納する方法を示す。本手法では前処理で求められた、点Oから交点Qまでの深さhを基準面の法線方向に直交するデプスマップに格納する。この時、格納する場所は点Oからデプスマップに垂線を下ろした点O'である。これは、視線方向に直交する面でデプスマップを計算した時よりも高いサンプリングレートでデータを持つためである。また、デプスマップにアクセスする時、点Oの基準面方向の場所 (テクスチャ座標) でデータにアクセスでき、座標変換の計算コストを減らすことができる。図 4(右) は複数のサンプリング点での前処理の結果、生成されたデプスマップの例である。

3.2 デプスマップを用いたレンダリング時の交点計算

本手法ではデプスマップからの深さ情報を用い交点の近似解を求める。そして、その近似点からレイキャスティングを行うことによって高速化を図る。図 5 に近似解の求め方を示す。

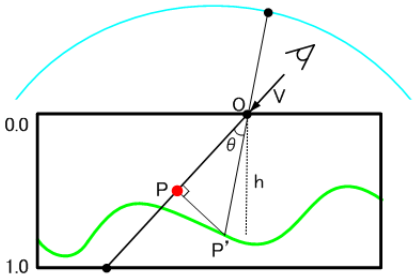

図 5 深さ情報からの近似解の求め方

まず初めに、視線ベクトルVを接線空間のベクトルに変換した上で、そのベクトルと最も方向が近いサンプリング方向を探す。これはVを極座標(θ, ϕ)に変換し、サンプリング点を分割した一定の角度で割ることで求められる。この時、サンプル点は視点から最も手前にある凹凸と交差することを保証するため、なるべく頂上点に近いサンプリング点を優先して選ぶ。次に選んだサンプリング点に対応したデプスマップを選び、O点でのテクスチャ座標を用いて格納された深さ情報を取得する。この深さ情報は点Oを通り、サンプリング方向を持つ半直線と凹凸との交点 (P') の深さである。我々は点Oを通り、視線ベクトル V の方向を持つ半直線と凹凸の交点の近似解を求めるため、点P'を視線ベクトルVに投影する。近似点までのベクトル OP は以下の式によって求める。

$$OP = |OP'| \cos\theta \frac{V}{|V|} \tag{1}$$

この時、θはベクトルVとベクトルOP'のなす角である。ベクトル OP を求めた後は近似点Pからレイキャスティングを行い、正しい交点を求める。この時、求められる近似点Pは二つのパターンに分けられる。一つは交点より手前にある場合である。そして、もう一つはすでに凹凸に潜り込んでいる場合である。Pが交点より手前にある場合はそのままレイキャスティングを行い、交点を求める。また、既に凹凸に潜り込んで

いる場合は視線を逆に戻り、外に出る方向に視線を進め、正しい交点を求める。

3.3 使用メモリの削減

本手法の前処理で作成されたデプスマップ群はマップのサイズによってはレンダリング時に多くのメモリを必要とする。また、サンプリングレートの低い、小さいサイズでデプスマップを作成すると、図 6 で示される赤い線の幅が、デプスマップの 1 ピクセルに相当し、その範囲の深さが同じになってしまう。そして、誤った深さで近似点を計算すると、近似点Pが凹凸を飛びえ超えてしまい、後ろの面が表示されてしまう場合がある。そこで本項では近似解の精度を考慮しながら効率よくデータを減らし、サンプリングレートが低い状態でもエラーを起さない手法を提案する。

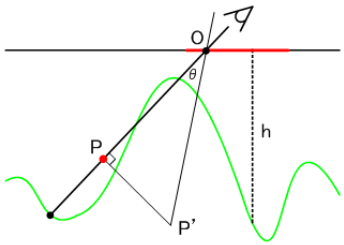

図 6 誤った近似点を求めた時の様子

まず、オーバーサンプリングでデプスマップを作成し、式 (2) の処理をデプスマップに対して行う。この時、tは元のデプスマップ、t'が処理後のデプスマップである。

$$t'[x,y] \leftarrow min(min(t[2x,2y],t[2x,2y+1]), \\ min(t[2x+1,2y],t[2x+1,2y+1])) \quad (2)$$

この式でダウンスケーリングを行い、デプスマップの深さはなるべく浅い物を残すことによってサンプリングレートの低い状態でも比較的正確な近似点を見つけることを可能とした。この処理の様子を図 7 に示す。5 節ではデプスマップのサイズを本提案法で削減しても近似解の精度の変化が少ない事を示す。

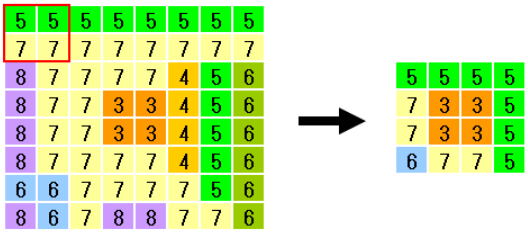

図 7 小さい値を残しながら、データを削減する処理

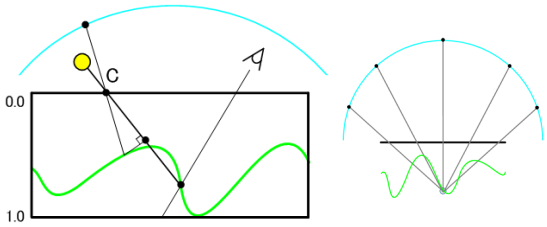

図 8 影の生成方法 (左) とAOでの遮蔽率の求め方 (右)

4. 影とアンビエントオクルージョン

この節では本手法により生成されたデプスマップ群を利用して、凹凸の作る影や凹凸によるアンビエントオクルージョン効果を計算する方法を示す。

4.1 影

写実的な表現を達成するのに影の効果は大きい。凹凸の表現においても、その凹凸が作る局所的な影を加えることは重要な項目である。凹凸が作る影の研究としてはホライズンマッピング[4] が有名である。我々は従来法[6] での影の生成に対して、提案した高速化が適応できる事を示す。

影を生成することは光源からの物体を見た時の遮蔽問題に繋がる[8]。そのため、視線から凹凸を表現するのと同様の手順で影を生成することができる。図 8(左) に示すように、光源からレリーフマッピングによって求められた交点が見えるかどうかで影を判定する。ここでは視線ではなく、光線を用いることで光点からの可視化判定を行う。まず、光点から凹凸が描画された点で光線ベクトルを作る。そして、そのベクトルと表面の交点を求め、そこを基点Cとする。続いて節 3 で提案した手法で近似点を求め、その点から交点に向かってレイキャスティングを行う。そして、光線との交点が視線との交点と一致するかを判定し、一致していなければ、そこは光点から見えない点として影の処理を行う。

4.2 アンビエントオクルージョン

アンビエントオクルージョン (AO) は大域照明を近似的に表現するための手法である。周囲の遮蔽の割合を求め、環境光の強さに反映することによって柔らかい陰影を表現することができる。

AO では obscurance 照明モデル[9] を特殊な条件のみを扱うことによって単純化している。環境光A_pは平面上のある点pと法線Nとすると、pを囲む半球Ωを可視化関数に従って積分することによって求められる:

$$A_p = \frac{1}{\pi}\int_{\Omega} V_{p,\omega}(N\cdot\omega)d\omega \quad (3)$$

この時、$V_{p,\omega}$はω方向に遮蔽物がない時に0を返す可

視化関数であり、積分によって遮蔽率が求められる。我々はこの式における可視化関数を、影の生成時と同様にデプスマップによる可視化判定に置き換えることで凹凸での遮蔽率を求めた。そして、図 8 に示すように交点での環境光を求めるため、半球の積分をサンプリング点の範囲で計算した。サンプリング点からの可視化判定をする時、本提案手法で求める近似解はサンプリング点からのベクトルと一致するため、非常に精度が高いものになり、より少ない回数で交点を求められる。この手法はシーン全体を考慮した大域照明とは違い、高さマップの凹凸による間接光しか影響しないが、大域照明特有の柔らかい陰影表現が可能となった。

5. 結　　果

三つの観点から本提案手法の評価を行った。計算環境は PentiumM 1.6GHz,1.52GB メモリ。実装は Java 言語で行い、JVM のバージョンは 1.5.0 である。また、全ての評価は、半球をθ方向に 32 分割、ϕ方向に 16 分割したサンプリング点を用いた。

一つ目の評価用のデータとしては球体に図 1(左) に示した高さマップをタイル状に貼り付けた物を使用した。球体に連続して並べることによって多くの角度からの高さマップの参照が行われる。なお、デプスマップのサイズは 64×64 の物を使用した。図 9 は提案法による描画結果と交点に着くまでの交差判定の回数を視覚化したものである。交差判定数が多いほど白くなっている。図から従来法より提案法の方が全体的に黒く、交差判定数が少なっていることが分かる。この画像では、従来法での交差判定の回数を約 51% 削減することに成功した。また、描画時間は従来法で 7.89 秒、提案法で 5.44 秒と 1.45 倍の高速化を達成した。

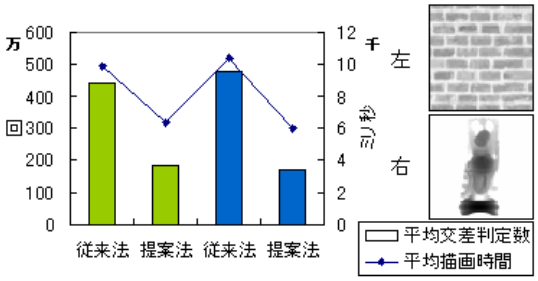

図 10　高さマップの違いによる交差判定と描画時間

次に貼り付けた高さマップの種類の違いによる、交差判定数と描画時間を図 10 に示す。なお、デプスマップは 32×32 の物を使用した。この評価では平面に 2 種類の高さマップを貼り付け、左から真ん中、右へと視点を移した。グラフは一枚あたりの平均描画時間と

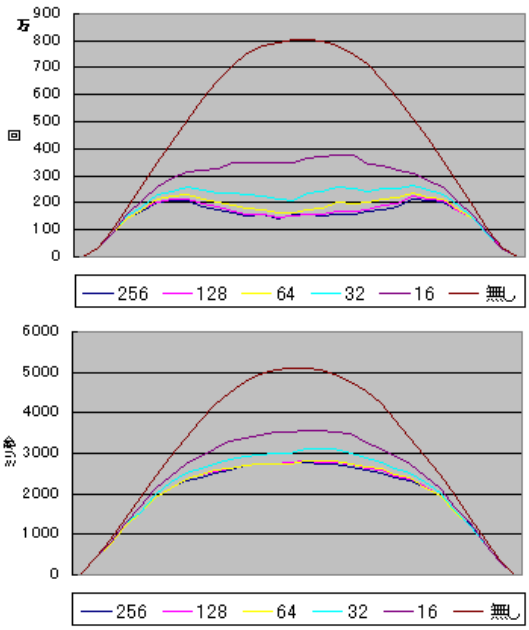

図 11　デプスマップのサイズの違いによるマップへのアクセス回数 (上) と描画時間 (下)

交差判定数の結果である。このグラフを見ると 2 種類とも従来手法よりも交差判定数が減っていることが分かる。そして、描画速度は従来法に比べて左の値では 1.55 倍、右の値では 1.74 倍高速化された。特に全体的に高さマップの高さが底に近い値で埋められている右の評価では、より高速化されている。

次にデプスマップのサイズの違いによる描画時間とデプスマップへのアクセス回数の評価を行う。平面に図 1(左) の高さマップを貼り付け、視線を左から右へと移した時の結果を図 11 に示す。なお、デプスマップは 512×512 のデプスマップを本提案手法で削減した物を使用した。上のグラフではバイリニアフィルタリングを行った時のデプスマップへのアクセス回数である。デプスマップのサイズがどの大きさであっても、アクセス回数が従来法に比べて、50%以下に抑えられている。また、サイズの違いによる回数の変化が少ないことが分かる。次に、下のグラフは実際の描画時間を比べたものだが、全体的にどのサイズにおいても高速化されている。特に平面と視線が直行する時に大きく高速化されていて、最高で 1.8 倍の高速化が実現している。なお、マップのサイズが 32 ピクセルの場合は平均して約 1.5 倍の高速化され、その時に必要なデプスマップ群のメモリ量は 192KB である。これは 256×256 のテクスチャが 1 枚増えたのと変わらない。

以上のことから本手法では以下の状況において特に高速化の効果があることが分かった。

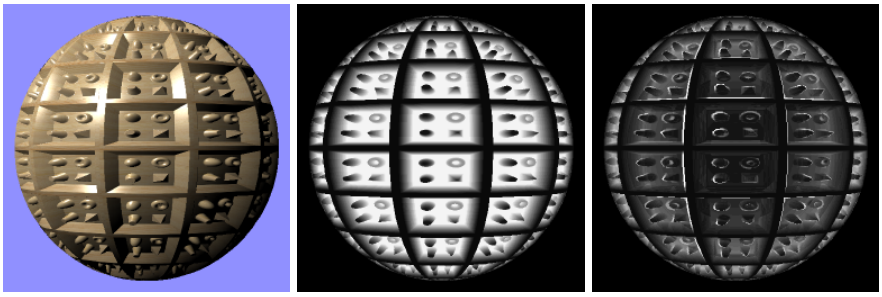

図 9 提案法による描画結果と交点までの交差判定の回数を可視化した結果、真ん中が従来法、右が提案法の結果である。

- 一回の交差判定に時間がかかる時
- 高さマップの値が全体的に底に近い時
- 視線が法線に近い時

最後に、本提案法のレリーフマッピングよって生成された画像に影とＡＯを加えた描画結果を図 12 に示す。これは Teapot に HappyBuddha の高さマップを貼り付けた画像である。影に加えて、柔らかい陰影表現を実現している。

図 12 レリーフマッピングに影とＡＯを加えた描画結果

6. ま と め

本稿ではデプスマップを用い、レリーフマッピングを高速化する手法を提案した。本手法では前処理で作成されたデプスマップの深さ情報を用いて近似解を求め、そこからレイキャスティングを行うことによって交差判定の回数を減らした。それにより高さマップへのメモリアクセスを減らし、レリーフマッピングの高速化を実現した。また、前処理で作成されたデプスマップ群のデータサイズを、近似解の精度を考慮しながら減らせることを示した。そして、前処理に使われたデプスマップで影や AO を計算できることを示した。今後の課題として曲面での正しいシルエットを求めるレリーフマッピング[5] に対応することが考えられる。

謝辞 本論文を作成するにあたり東京大学大学院の海野広志氏に多大なる助言および文章の校正をしていただきました。ここに感謝の意を表します。

参 考 文 献

1) Blinn, J. F. “Simulation of wrinkled surfaces,” *In Proceedings of the 5th annual conference on Computer graphics and interactive techniques,* ACM Press, 286-292, 1978.
2) CooK, R. L. “Shade trees,” *In Proceedings of the 11th annual conference on Computer graphics and interactive techniques,* ACM Press, 223-231, 1984.
3) Kaneko, T., Takahei, T., Inami, M., Kawakami, N., Yanagida, Y., Maeda, T.,and Tachi S. “Detailed shape representation with parallax mapping,” *In Proceedings of the ICAT 2001,* 205-208, 2001.
4) Max, N. “Horizon mapping: shadows for bump-mapped surfaces.” *The Visual Computer 4, 2, 109-117,* 1988.
5) Oliveira, M. M.,and Policarpo, F. “An Efficient Representation for Surface Details,” *UFRGS Technical Report RP-351,* 2005.
6) Policarpo, F., Oliveira, M. M.,and Comba, J. “Real-Time Relief Mapping on Arbitrary Polygonal Surfaces,” *ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games,* Washington, DC, 155-162, 2005.
7) Wang, L., Wang, X., Tong, X., Lin, S., Hu, S., Guo, B.,and Shum, H.Y. “View-Dependent Displacement Mapping,” *ACM Trans. Graphics (Proc. SIGGRAPH) 22,* 3, 334-339, 2003.
8) Williams, L. “Casting curved shadows on curved surfaces,” *In Siggraph 1978, Computer Graphics Proceedings,* 270-274, 1978.
9) Zhukov, S., Iones, A.,and Kronin, G. “An ambient light illumination model,” *In Rendering Techniques ’98 (Proceedings of the Eurographics Workshop on Rendering),* 45-55. 1998.