3角形再分割による3次元モデルへの情報埋め込み

4E－6－3

柴 誠，茅 暁陽，今宮 淳美
山梨大学工学部 コンピュータメディア工学科

1．はじめに

近年，PC やインターネット，そして各種入出力機器の普及に伴い，画像，文字，音声，またはそれらを相互に組み合わせたデジタルメディアに対する，加工や複製，公開が容易に行えるようになった．しかしそれは同時に，権利が無い者であっても容易に無断使用できるという危険性を内包している．つまり，苦労して創作したコンテンツが，著作者の知らない間に第三者が創作したように加工され，公開される恐れがある．このような事を未然に防ぐために，電子透かしをコンテンツ自身に埋めこむことで，著作権表示を行えるようにする技術が注目されている．本論文では3次元ポリゴンデータに対する新しい透かし埋め込み技術を提案する．

3次元ポリゴンデータは，仮想空間の構築やＣＧアニメーションの製作などに必要不可欠なコンテンツである．現在までに，2次元画像や音声，電子文書などにおける電子透かし技術は数多く開発されている．3次元ポリゴンデータに関しては，９８年に大渕らによってはじめて提案され，その後いくつかの方法が提案されている[1,2,3,4]．本論文で提案する手法は，三角形の辺を新しい頂点で2分割し，線分の長さの比に情報を埋め込む．同一直線上にある2本の線分の長さの比がアフィン変換に対して保存されることから，本手法により埋め込まれた情報は，アフィン変換に耐えることができる．また一辺につき，最大8バイトのデータ埋め込みが可能という大容量性，幾何学的誤差を生じない，透かしの抽出に元のモデルデータを必要としないという利点を兼ね備えている．

第2節では電子透かしについて簡単に説明し，関連研究を紹介する．第3節では今回提案する手法の詳細を説明していく．第4節で本手法の特徴と問題点について考察する．第5節で今後の課題を示し本論文をまとめる．

2．関連研究

電子透かし技術は次に挙げるような要求を満たす必要がある．

1）流通による複製が行われた場合も透かし情報は保存される
自由な流通を妨げないよう，保存しなおしたり複製したりしたときに，フォーマットの変更やヘッダ部の書き換えが行われても透かし情報が失われないように，データそのものに埋めこむ必要がある．

2）透かし情報は通常使用に耐えなくてはならない
データを変形したり，一部を切りとって使用したりする場合も透かし情報が保存される必要がある．

3）透かし情報は必要な場合，容易に復元できる
透かし情報は容易に参照できなくてはならない．必要に応じてユーザが参照することで，著作者などの情報を提示する．

4）透かし情報の長さはそのモデルを識別できるぐらいの長さが必要
著作権情報やモデルの情報を埋めこむので，少なくとも数十ビットの情報が埋めこめられなければならない．また，こうして埋めこんだ情報をデータのあちこちに繰り返し配置できるようなアルゴリズムが望ましい．

Digital Watermarking for 3D Polygons using Retriangulation.

Makoto Shiba , Xiaoyang Mao , Imamiya Atsumi
Department of Computer and Media Engineering , Yamanashi University

5）情報を埋めこむことで起こる誤差は使用に耐える程度でなくてはならない
データ自身に情報を埋めこむことで，誤差が生じる場合は，できるだけ誤差を小さくし，通常表示でその誤差が認められない程度でなくてはならない．

これまでに提案されている透かし技術は，オブジェクト空間方法と周波数空間方法の二つに大別される[5]．オブジェクト空間方法は画像や３次元モデルに直接変更を加え，情報を埋め込むのに対して，周波数空間方法はフーリエ変換やウェーブレット変換により，データを一度，周波数空間に変換し，周波数成分やウェーブレット係数に情報を埋め込む．

３次元ポリゴンデータに関するオブジェクト空間方法は，９８年に大渕らによって提案された[1]．彼らは点，線，多面体などからなる広い意味での３次元ポリゴンモデルを対象とし，その頂点座標値，頂点間の接続関係またはその両方を変更することにより情報を埋め込む５つのアルゴリズムを提案した．そのうち，アフィン変換に耐えるものとして，四面体の体積比を利用したアルゴリズムと，帯状の三角形の接続を利用したアルゴリズムがある．四面体の体積比を利用したアルゴリズムでは，初期点を決定し，そこから延びる三角形の列，そしてそれらの三角形を含む四面体の列を作る．次に，最初の四面体の体積に対するその他の四面体体積の比に情報を埋め込む．三角帯のアルゴリズムでは，トライアングルストリップに対して情報を埋め込む．初期辺を決定し，その辺を含む三角形の残り２辺に，それぞれ"0"と"1"と割り当て，ビット列に対応する辺へ接続する三角形を辿っていくことで情報を埋め込む．このアルゴリズムは１モデルあたりに埋め込むことのできる情報量は少なく，空間効率の面において問題がある．一方，伊達ら [2]は周波数空間方法として，３次元ポリゴンデータにウェーブレット変換を施し，ウェーブレット係数に情報を埋め込む手法を開発した．この手法で埋めこまれた情報もアフィン変換に耐えることができる．また幾何学的誤差を容易にコントロールできるという利点も持つ．しかし，抽出には元モデルが必要であり，またウェーブレット変換と逆変換に多くの処理時間を要する．

３．本論文で提案する手法

３－１．概要

本論文では，三角形メッシュで表されている３次元モデルを対象とし，三角形の辺を新しい頂点で２分割し，線分の長さの比に情報を埋め込む新たな手法を提案する．図 3.1 に示すように，モデルデータを構成する三角形の１辺に新たな点を打ち，元の辺の長さ b と，辺の１つの端点から追加した点までの長さ a の比 a / b が埋め込む情報となるように新しく加える点の座標値を決定する．そして三角形の３辺に打った点を結び，元の三角形を４つに分割する．これは情報を埋め込んだ後のデータも元のデータと同じ形状を表す正しいポリゴンメッシュとするためである．同一直線上にある２本の線分の長さの比がアフィン変換に対して保存されることから，本手法により埋め込まれた情報は，アフィン変換に耐えることができる．

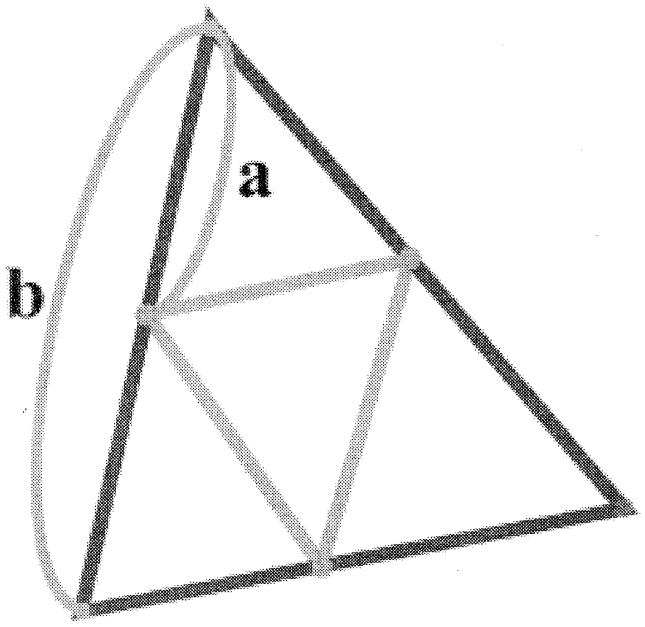

図 3.1 三角形の辺に新しい点を加え，三角形を 4 つに分割する

３－２．埋め込みアルゴリズム

アルゴリズムへの入力は３次元モデルの三角形メッシュ，埋め込むべき情報のストリング，および擬似乱数を発生させるためのシードである．擬似乱数は三角形の辺を辿る順序を決定す

る二つの開始マーカと，ストリングの終端を示す終了マーカおよび，埋め込みをはじめる三角形の決定に使用される．前処理として，元のモデルにおいて3点が直線上にある場合，中間にある点の位置に微小な変更を加える．これにより，埋め込んだ情報を抽出するときには，同じ直線上にある3点を検出することにより，情報が埋め込まれている辺を抽出することが可能になる．

図 3.2 を用いて埋め込み手順を概説する．まず，入力したシードをもとに乱数を生成し，最初に埋め込む三角形を決定する．続いて擬似乱数を二つ生成し，これらを順序を決定する開始マーカとしてその三角形の2辺に埋め込む（①，②）．そして，3つ目の辺にユーザーから与えられた入力文字列の1文字目を埋め込む（③）．新たに加えた3点を相互に接続し，三角形を4つに分割する．残りの文字列を埋め込むため，辿ってきた最後の辺に接続する三角形を探索し，最初の三角形と同様に2辺に情報を埋め込む（③～⑦）．もし，この三角形が見つからなかった場合，埋め込みに使用したひとつ前の辺に戻り，同様の処理を行う（⑧以降）．最後に，埋め込んだ箇所を初期点を残し元モデルから切り離す(図 3.3)．

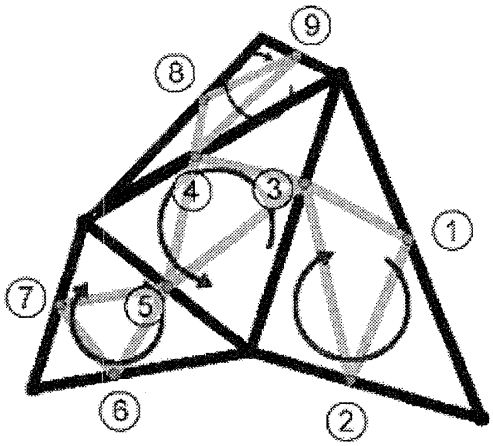

図 3.2 埋め込み手順

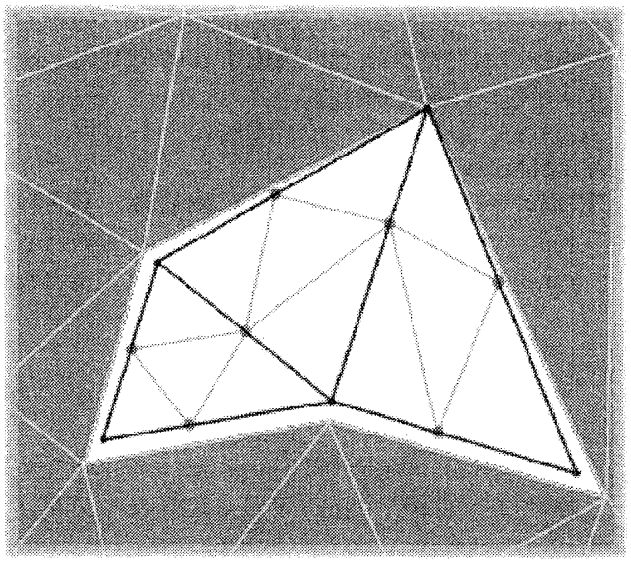

図 3.3 元モデルから埋め込みが行われた部分を切り離す

以下に，アルゴリズムの擬似コードを示す．

ポリゴンメッシュの頂点を座標のセット <x,y,z> ，辺を頂点のセット< vertex1, vertex2 >，三角形を頂点のセット< vertex1, vertex2, vertex3>として表すとする．

入力

in_triangle_list：3次元モデルの三角形リスト

msg：入力文字列

seed：乱数生成用のシード

出力

out_triangle_list：情報が埋め込まれている3次元モデルの三角形リスト

作業用変数

M1,M2：seed から生成される開始マーカ

M3：seed から生成される終了マーカ

start_vertex：埋め込みを開始した点

symbol：1辺に埋め込むシンボル

vertex1～vertex 3：これから埋め込もうとする三角形の各頂点

new_vertex1～new_vertex3：埋め込み処理により追加された点

line_buffer：情報を埋め込んだ辺を格納するバッファ，各要素は辺の両端点と追加した新しい頂点のセット <vertex1,embeded_point,vertex2 >として表される．

embeded_ triangle_list：情報が埋め込まれた三角形のリスト
unembeded_ triangle_list：情報が埋め込まれていない三角形のリスト

アルゴリズム

```
embed( in_triangle_list , msg)/* モデルデータ in_triangle_list に文字情報 msg を埋め込む */
{
        M1 = rand(seed); M2 = rand(seed); M3 = rand(seed);
        msg += M3;
        unembeded_ triangle_list   =   in_triangle_list;
        k=0;

        vertex1 = in_triangle_list[rand(seed)]. vertex 1;        /* 埋め込みを始める三角形を決定する */
        vertex2 = in_triangle_list[rand(seed)]. vertex 2;
        vertex3 = in_triangle_list[rand(seed)]. vertex 3;
        start_vertex = vertex 1;

        symbol = next(msg);
        new_ vertex1 = embed_point (vertex 1, vertex 2, M1);  /* M1,M2 とシンボルを埋め込む */
        new_ vertex2 = embed_point (vertex 2, vertex 3, M2);
        new_ vertex3 = embed_point (vertex 3, vertex 1, symbol);

        symbol = next(msg);
        line_buffer[k] += <vertex 3, new_ vertex3, vertex 1>; /* シンボルを埋め込んだ辺を line_buffer に
加える */
        embeded_triangle_list   += < vertex 1, vertex 2, vertex 3>;
        unembeded_triangle_list -= < vertex 1, vertex 2, vertex 3>;

        /* 三角形を 4 つに分割する */
        out_triangle_list += < new_vertex 1, new_vertex 2, new_vertex 3 >;
        out_triangle_list += < vertex 1, vertex 2, new_vertex 1 >;
        out_triangle_list += < new_vertex2, vertex 2, vertex 3 >;
        out_triangle_list += < vertex 1, new_vertex3, vertex 3 >;

        symbol = next(msg);
        do{     /* 入力文字情報が終わるまで以下のように埋め込みを繰り返す */
                vertex 2 = vertex 1; vertex 1 = vertex 3;
                vertex3 = next_triangle(vertex 1,vertex 2, unembeded_ triangle_list);
                        /* 次に辿る三角形が見つからなかった場合，バッファ内の一つ前の辺に戻る */
                while( vertex3 = next_triangle(vertex 1,  vertex 2) = FALSE && k > 0 )
                        <vertex1, new_vertex1,vertex2 > = line_buf[--k];
                if( k = 0 )print("Embedding capasity for this model is full !! ");break;

                /* 次の二つのシンボルを埋め込む */
                symbol = next(msg);
                new_ vertex2 = embed_point(vertex 2, vertex 3 , symbol);
                line_buffer [++k]= < vertex 2,new_vertex2, vertex 3 >;

                if( symbol != M3 ) symbol = next(msg);
```

```
                new_ vertex3 = embed_point (vertex 3, vertex 1,symbol);
                line_buf [++k]= < node3 , new_ vertex3, node1 >;
                embeded_ triangle_list += < vertex 1, vertex 2, vertex 3 >;
                unembeded_ triangle_list -= < vertex 1, vertex 2, vertex 3 >;

                out_triangle_list += < new_vertex 1, new_vertex 2, new_vertex 3 >;
                out_triangle_list += < vertex 1, vertex 2, new_vertex 1 >;
                out_triangle_list += < new_vertex2, vertex 2, vertex 3 >;
                out_triangle_list += < vertex 1, new_vertex3, vertex 3 >;
        } while( symbol != M3 );
        out_triangle_list += unembeded_ triangle_list;
}

embed_point(node1, node2, symbol) /* 辺<node1,node2>に symbol を埋め込む */
{
        ratio = msg_to_float(symbol);

        x = ((node2.x - node1.x) * ratio) + node1.x;      /* 新しい頂点座標の計算 */
        y = ((node2.y - node1.y) * ratio) + node1.y;
        z = ((node2.z - node1.z) * ratio) + node1.z;

        return <x,y,z>;
}

next(msg)       /* msg から埋め込むシンボルを取り出す */
msg_to_float    /* キャラクタデータを数値に変換する */
next_triangle(vertex1,vertex2,unembeded_triangle_list) /* vertex1,vertex2 を二つの頂点とする三角形を
検索し，第３の頂点を返す */
```

３－３．抽出アルゴリズム

まず，同一直線上にあり，一つの端点を共有する辺を検出する．これを埋め込みの行われた元のモデルの一辺とみなし，このような辺のリストを生成する．そして，このリストを用いて埋め込みの行われた三角形のリストを生成する．次に開始マーカーが埋め込まれた２つの辺を探索し，これを二辺とする三角形のもう一つの辺から最初のデータを抽出する．以降埋め込み時と同様の順序で三角形の辺を辿っていき，終了マーカが抽出されるまで処理を続ける．

以下に，抽出アルゴリズムの擬似コードを示す．

埋め込みのときと同様，ポリゴンメッシュの頂点は座標のセット <x,y,z> ，辺を頂点のセット< vertex1, vertex2 >，三角形を頂点のセット< vertex1, vertex2, vertex3>とする．

入力

embeded_triangle_list：埋め込みにより更新された３次元モデルの三角形リスト

seed：乱数生成用のシード（埋め込み時に使われたものと同じ）

出力

out_msg：抽出した文字列情報

作業用変数

tmp_line_list：情報の埋め込まれた部分の辺のリストで各要素は，端点と埋め込み時に追加した頂点のセット<vertex1,embeded_point,vertex2 >として表される．

tmp_triangle_list：情報の埋め込まれた部分の三角形リスト

M1,M2：seed から生成される開始マーカ
M3：seed から生成される終了マーカ
vertex1～vertex3：これから抽出しようとする三角形の各頂点
embeded_point：情報の埋め込み時に追加した点
line_buffer：情報を埋め込んだ辺を格納するバッファ<vertex1,embeded_point,vertex2 >で表す.

アルゴリズム

```
extract( embeded_triangle_list , seed) /* 埋め込まれた三角形リストとシードから文字情報を抽出する
*/
{
        M1 = rand(seed); M2 = rand(seed); M3 = rand(seed);
        tmp_line_list = recover_line(embeded_triangle_list);
        tmp_triangle_list = recover_triangle(tmp_line_list);
        k=0;

        /* tmp_line_list から M1,M2 が埋め込まれている辺を探す */
        for( i = 0 ; tmp_line_list[i] ! = NULL ; i++){
                symbol=extract_point(tmp_line_list[i].vertex1,tmp_line_list[i].embed_point,tmp_line_list
        [i].vertex2);
                if(symbol == M1){
                        vertex1 = tmp_line_list[i].vertex1;
                        vertex2 = tmp_line_list[i].vertex2;
                }
                if( symbol == M2){
                        vertex2 = tmp_line_list[i].vertex1;
                        vertex3 = tmp_line_list[i].vertex2;
                }
        }

        /* 最初の文字列情報を抽出する */
        embeded_point = added_point(vertex2,vertex3,tmp_line_list);
        symbol = extract_point(vertex1,embed_point ,vertex2);
        out_msg += symbol;
        line_buffer[k] += <vertex3,embed_point,vertex1>;

        if( symbol != M3 ){       /* M3 を抽出されるまで以下を繰り返す */
                for( symbol != M3){
                        vertex2 = vertex1; vertex1 = vertex3;
                        /* 次に辿る三角形が見つからなかった場合，バッファ内の一つ前の辺に戻る */
                        while( vertex3 = next_triangle(vertex 1, vertex 2 ,tmp_triangle_list) = FALSE &&
k > 0 )<vertex1, vertex2, vertex3 > = line_buf[--k];
                        /* 次の二つのシンボルを抽出する */
                        embeded_point = added_point(vertex2,vertex3,tmp_line_list);
                        symbol = extract_point(vertex1,embed_point ,vertex2);
                        out_msg += symbol;
                        line_buf[++k] += <vertex2,embed_point,vertex3>;
                        if( symbol == M3 )return out_msg;

                        embeded_point = added_point(vertex2,vertex3,tmp_line_list);
```

```
                    symbol = extract_point(vertex3,embed_point,vertex1);
                    out_msg += symbol;
                    line_buf[++k] += <vertex3,embed_point,vertex1>;
                }
        }

return out_msg;
} End of extract

/* node1,node2 と埋め込み時に追加した頂点 embed_point から埋め込まれた情報を出力する */
extract_point(int node1,enbed_point, int node2)
{
        ratio = |<node1 , emb_node >| / |<node1 , node2>|;
        out_msg = float_to_msg(ratio);
        return out_msg;
}
recover_line(embeded_triangle_list)    /* 情報が埋め込まれている箇所の辺のリストを生成する */
recover_triangle_list(tmp_line_list)    /* 情報が埋め込まれている箇所の三角形のリストを生成する */
next_point(vertex1, vertex2)    /* vertex1, vertex2 を端点とする辺の間にある点を返す */
float_to_msg    /* 実数をキャラクタデータに変換する */
next_triangle(vertex1,vertex2) /* vertex,と vertex を二つの頂点とする三角形を検索し，もう一つの頂点を返す */
```

4．考察

前節で述べたアルゴリズムを Windows 上で実装し，いくつかの３次元モデルへの用した．モデルデータに文字情報“this is test message.”を埋め込んだ様子を図 4.1 に示す．白い線で描かれているポリゴンが情報の埋め込まれている部分である．

今回の実装では１つの辺に対し１つのシンボルを埋め込んだが，倍精度実数を用いることで，原理的には三角形の１辺につき８バイトの情報を埋め込むことができる．したがって既存の方法と比べ空間効率は格段に良いといえる．また，元の３次元モデルに対して幾何学的誤差をいっさいもたらさないことや，埋め込みを行った箇所が目立たないこと，抽出に元モデルを必要としないことも大きな利点としてあげられる．ただし，埋め込まれた情報を正確に抽出できることを保証するために，元のモデルにおいて３点が直線上に並んでいる場合，中間にある点に位置を微小な変更を加えなければならず，この処理により幾何学的な誤差が生じる．また，モデルに対する幾何変換などによる頂点座標の丸め誤差や，頂点座標に加わるノイズの影響を考慮した場合，演算において，誤差に対する一定の許容範囲を持たせる必要がある．点の位置に加わる変更は，この範囲を上回る必要がある．本手法は，ポリゴンメッシュのトポロジを利用するため，埋め込まれた情報は，モデルの部分変形や

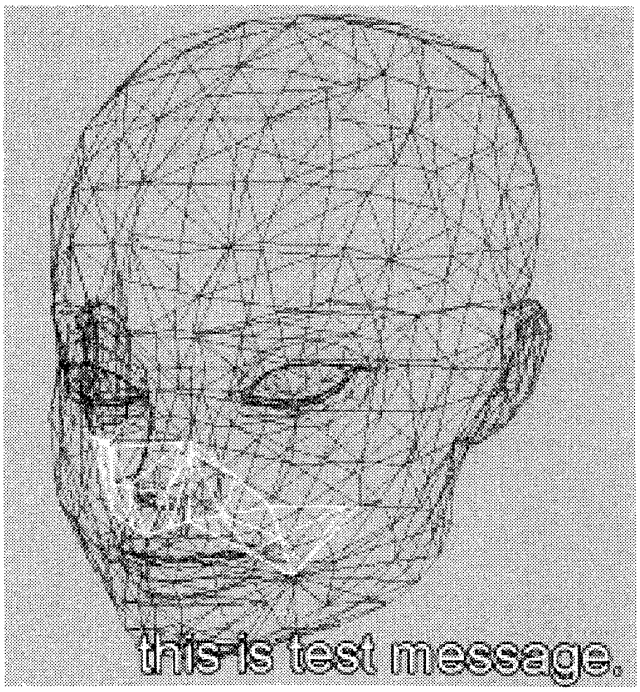

図 4.1 埋め込みを行ったモデルデータ

切断により破壊される．しかし埋め込み処理は3次元モデルの局所で行われるので，空間効率が極めて高いという利点を生かし，モデルの複数箇所に情報を反復して埋め込むことにより，切断や部分変形に対しての頑強性を増すことが出来る．一方，埋め込みに使われた三角形は4つの三角形に分割されるため，ポリゴン数が最大で元モデルの4倍に増加する事が問題として挙げられる．しかしこれに関しても，1つの三角形に対して埋め込むことのできる情報量が大きいため，ほとんどの応用において，埋め込みによる三角形の数の増加量を低く抑えることができると考えられる．今回の例ではポリゴン数 1349 のモデルに文字情報"this is test message"(21 文字)を埋め込みを行った．一つの辺に1キャラクタの情報を埋め込み，増加した三角形の数は 44 である．また，三角形の分割によりシェーディング時に色の不連続を生じる場合がある．この問題は，色の変化が小さい領域を埋め込み領域として選択することで，改善できると考えられる．

5．まとめ

本論文では3次元ポリゴンデータに対する新しい透かし埋め込み技術を提案した．第2節で述べた要求をすべて満たすような透かし技術の開発は困難であると思われるが，本論文で提案した方法は，有効な情報埋め込みの手法の一つとして，既存の手法とを組み合わせる事でインターネットやマルチメディアなどの応用分野において役立つことが期待される．

今後の課題として，幾何学的な誤差や抽出時の誤りを出来るだけ少なくするように，アルゴリズムを改善していくことがまず挙げられる．本手法において1つの三角形を分割することにより得られる4つの三角形は同一平面上にあるため，ポリゴン削減処理により簡単に1つの三角形に再統合される．ポリゴン削減処理をはじめ，様々な処理に対する頑強さを増すことも重要な課題の一つである．

6．参考文献

[1] Ryutarou Ohbuchi, Hiroshi Masuda, and Masaki Aono, "Watermarking Three-Dimensional Polygonal Models", *IEEE Journal on Selected Areas in Communications*. Vol. 16,No. 4, pp.551-560,May 1998

[2] S. Kanai, H. Date, and T. Kishinami, "Digital Watermarking for 3D Polygons using Multiresolution Wavelet Decomposition", *Proc. of the Sixth IFIP WG 5.2 International Workshop on Geometric Modeling: Fundamentals and Applications (GEO-6)*, pp. 296-307, Tokyo, Japan, December 1998.

[3] O. Benedens, "Geometry-Based Watermarking of 3D Models", *IEEE CG&A*, pp. 46-55, January/February 1999.

[4] Emil Praun, Hugues Hoppe, Adam Finkelstein,"Robust Mesh Watermarking", *MSR-TR-99-05, Microsoft Research*, 1999. (To be published at SIGGRAPH '99)

[5] 松井甲子雄, "電子透かしの基礎" ,森北出版社,1998