ポリゴンデータ幾何圧縮のための局所変換の実装方式

長野　寛[†]（正員）　客野　一樹[†]（正員）
松本　建太[††]（正員）　徳永　隆治[†]（正員）

Implementation of Local Transformations for Geometric Compression for Polygon Data

Hiroshi NAGANO[†], Kazuki KYAKUNO[†], Kenta MATSUMOTO[††], and Ryuji TOKUNAGA[†], *Members*

[†] 筑波大学システム情報工学研究科コンピュータサイセンス専攻，つくば市
Graduate School of Systems and Information Engineering, Department of Computer Science, University of Tsukuba, 1–1–1 Tennodai, Tsukuba-shi, 305–8573 Japan

[††] (株) アクセル，東京都
Axell Corporation, 4–14–1 Sotokanda, Chiyoda-ku, Tokyo, 101–8973 Japan

あらまし　ポリゴンデータのひずみ幾何圧縮の中で，隣接する三角形から生成された基底系による局所変換符号化は一つの効率的な方式である．これらは，基底ベクトルの算出に平方根と除算を含む正規化処理を必要とするため，算術演算量の観点から改良の必要性がある．本研究は，平方根の逆数を関数テーブルとビットシフトで置換する方式を提案し，その有効性を検証する．

キーワード　ポリゴンモデル，幾何圧縮，局所直交変換，高速化

1. まえがき

3D グラフィックス，特にリアルタイムレンダリングにおいて，ポリゴンは重要かつ一般的な方法論である．その表現の多様性と数学的枠組みの利便性から，科学技術計算における可視化，CAD・CAM の工業分野のみならず，コンピュータゲーム等のアミューズメントに広く応用されている．加えてポリゴンは，OpenGL や DirectX 等の 3D グラフィックス API における中心的手法であり，世界中のグラフィックスハードウェアに最も多く実装されている．近年，コンピュータアミューズメント分野における 3D グラフィックスの発展は特に著しく，グラフィック専用プロセッサの普及や OpenGL ES 等の組み込み機器向け API の策定にも後押しされ，携帯ゲーム機や携帯電話等の小型端末でも大量のポリゴンを用いたコンテンツが増加している．しかし，これらの組込み機器においては，データ伝送用のバス幅は依然として狭く，パイプラインにおける伝送速度がポリゴンの高度利用の障害となっている．

この問題を解消するため，ポリゴンデータの転送コストを下げる様々な圧縮方式が検討されており，位相情報を保存する圧縮方式としては，

（1）　座標や法線に関する幾何圧縮 [1]～[7]

（2）　頂点の接続情報に関する位相圧縮 [1], [5], [6]

等がある.（2）はデータ損のない無ひずみ圧縮であるが,（1）は，量子化・予測符号化・エントロピー符号化等の要素技術の組合せで構成されており，無ひずみとひずみ圧縮がある．科学技術用データ及び CAD・CAM データに対しては，無ひずみ圧縮の適用が不可欠であるが，コンピュータアミューズメントに用いられるマルチメディアデータにおいては，次の 2 点から，ひずみ幾何圧縮が有効と考えられる．

(a)　画像データや音響信号と同様の主観的利用であるため，モデルにおける凹凸等の曲面上の微細構造は，テクスチャにより二次元画像として扱われる．このためポリゴンそのものには，曲面の連続性や滑らかさの幾何学的冗長性が多く含まれている．

(b)　ポリゴンデータのほとんどは浮動小数で記述される数値情報であり，無ひずみ圧縮では高い圧縮効率が得られない．

上記の観点から提案されたひずみ幾何圧縮の多くは，隣接三角形の情報を用いた予測符号化方式であり，特に平行四辺形予測 [1], [2] が一般的に用いられている．しかし，同方式は，データを大域座標系で一様に処理するため，予測残差の冗長性が 3 自由度に分散し，圧縮効率の劣化が生じていると考えられる．これに対して，文献 [3] あるいは [4] は，隣接三角形の情報から局所座標系を構成し，残差を局所変換して圧縮効率を高める方式を提案している．

上記方式において局所変換の計算が平方根の逆数を用いたノルムの正規化処理を含むことに注意を促す．汎用 CPU において除算及び平方根計算は極めて低速であり，例えば Intel Pentium4 における IA-32 命令であれば，加減算の 11 倍ものクロックサイクル数が必要である [8]．したがって，頂点ごとに基底系を算出する場合，正規化処理の負荷は無視できない．

そこで，本論文は，ノルムの二乗値を仮数部と指数部に分解し，仮数部に対する小規模な関数テーブルを用いることで，ノルムの正規化処理を乗算とビットシフトで実装する方式を提案する．また，計算機実験により平方根の逆数を用いた場合と処理速度及び精度を比較し，その有効性を検証する．

2. ポリゴンデータの局所変換符号化の背景

文献 [5] は，ポリゴンデータの一般的応用において

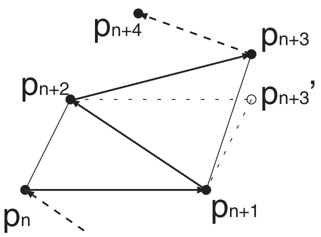

図 1　平行四辺形予測
Fig. 1　Paralellogram prediction.

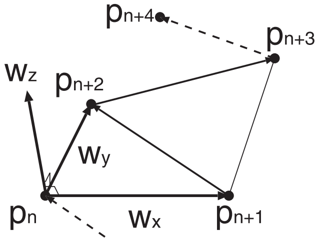

図 2　文献 [3] の局所基底系
Fig. 2　Local coordinate system in [3].

32 [bit] 浮動小数による頂点座標の記述が冗長であることを指摘し，整数精度への量子化が有効となることを報告している．更に，文献 [1], [2] は，全頂点を量子化した後，隣接する二つの三角形が平行四辺形をなすと仮定し，一定の順路で辺を巡りながら頂点 $\{p_n\}$ を差分ベクトルとして予測符号化する方式を提案している．例えば，図 1 における三角形 $\{p_n, p_{n+1}, p_{n+2}\}$ から p_{n+3} の予測値

$$p'_{n+3} = p_{n+2} + p_{n+1} - p_n \in R^3$$

を生成し，残差ベクトル $d = p_{n+3} - p'_{n+3} \in R^3$ を符号化対象とする．他方，文献 [3] は，隣接三角形 $\{p_n, p_{n+1}, p_{n+2}\}$ から非直交基底系

$$w_x = p_{n+1} - p_n \in R^3,$$
$$w_y = p_{n+2} - p_n \in R^3,$$
$$w_z = \frac{w_x \times w_y}{||w_x \times w_y||}\sqrt{||w_x||||w_y||} \in R^3 \qquad (1)$$

を構成し，差分ベクトル $d = p_{n+3} - p_n \in R^3$ を座標 (1) に関する 3 成分へ変換し，符号化対象とする方式を提案している．（図 2 参照）特に，符号化においては，誤差の累積を防ぐため，圧縮結果から p_{n+3} を逐次的に復号（伸張）し，これを更新した後に次の p_{n+4} を $\{p_{n+1}, p_{n+2}, p_{n+3}\}$ を用いて予測する必要がある．また，復号においても基底ベクトル w_z の計算に外積

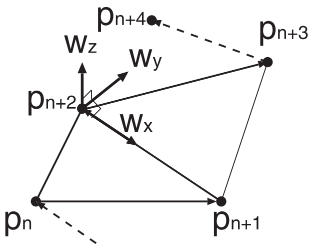

図 3　文献 [4] の局所基底系
Fig. 3　Local coordinate system in [4].

のノルムの逆数 $||w_x \times w_y||^{-1}$ が必要となることに注意する．更に，文献 [4] は，2 辺

$$x = p_{n+1} - p_{n+2} \in R^3,$$
$$y = p_{n+1} - p_n \in R^3$$

を順に正規直交化した基底系

$$w_x = \frac{x}{||x||} \in R^3,$$
$$w_y = \frac{y - (y \cdot w_x)w_x}{||y - (y \cdot w_x)w_x||} \in R^3,$$
$$w_z = w_x \times w_y \in R^3 \qquad (2)$$

を構成し，差分ベクトル $d = p_{n+3} - \frac{p_{n+1} + p_{n+2}}{2} \in R^3$ を座標 (2) に関する 3 成分へ変換し，符号化対象とする方式を提案している．（図 3 参照）この場合も，三角形と共面となる $\{w_x, w_y\}$ の正規化処理に逆数 $||x||^{-1}$ 及び $||y - (y \cdot w_x)w_x||^{-1}$ が必要となることに注意する．以降，ノルムの逆数を算術演算で求める正規化処理を通常方式と呼ぶ．

3. 提案方式

3.1 関数テーブル化

正規化対象となるベクトルのノルムの二乗値を 2 進小数で表現し仮数部 $m \in [1, 10)(= [2^0, 2^1))$ 及び指数 $\epsilon \in Z$ として $f = 2^\epsilon \times m$ とする．ここで，ϵ が奇数の場合は，$\epsilon = 2e + 1, M = 2m$，偶数の場合は $\epsilon = 2e, M = m$ とすると，ノルムの逆数は，

$$\frac{1}{\sqrt{f}} = 2^{-e} \times \frac{1}{\sqrt{M}} \qquad (3)$$

となる．ここで，$M \in [1, 100)(= [2^0, 2^2))$ の上位 n ビットを取り出し，整数 $\{0, 1, \ldots, 2^n - 1\}$ と対応づけると，関数台 $[1, 100)(= [2^0, 2^2))$ を 2^n 分割した関数テーブル

$$\left\{ \frac{1}{\sqrt{M_i}} : i = 0, 1, \ldots, 2^n - 1 \right\}$$

が得られる．よって，M の上位ビットによって関数値を引き，2^{-e} をビットシフトで実行することで平方根と除算を省くことができる．

3.2 実装方式

実数を IEEE754 形式で扱うものとして，以下のように提案方式を実装する．

（a） 関数テーブルの作成

以下，実数 f の指数部を $e(f)$，仮数部を $m(f)$ と表記し，図 4 のように指数部の下位 1 ビットから仮数部の上位 $n-1$ ビットまでの値を $I(f)$ とおく．また，関数テーブルの値を $\{T_i|i=0,1,\ldots,2^n-1\}$ とする．なお，$e(f)$ は実際の指数値に 127 のバイアスが加えられていることに注意する．

$M \in [1,100)(=[2^0,2^2))$ は指数部の下位 1 ビットと仮数部から決定されるため，

$$\{T_{I(M)} = M^{-\frac{1}{2}}|I(M)=0,1,\ldots,2^n-1\}$$

とすればよい．ただし，入力の丸め誤差を抑えるため，$m(M)$ の左から n ビット目は常に 1 とする．また，指数部はバイアスによって下位 1 ビットが反転するため，

$$e(M) = \begin{cases} 128 & (i < 2^{n-1}) \\ 127 & (i \geq 2^{n-1}) \end{cases}$$

とする．なお，$M^{-\frac{1}{2}} \in (2^{-1},2^0)$ から，T_i の指数部は 126 と一意に定まるため，テーブルにおいては仮数部の値のみを保持する．

（b） 正規化の処理

(3) において，e は ϵ の算術右シフトによって得られるので，$r \simeq \dfrac{1}{\sqrt{f}}$ の指数部と仮数部は，以下によって求められる．

$$e(r) = 126 - \{(e(f)-127) \gg 1\},$$
$$m(r) = m(T_{I(f)})$$

ただし，$\gg$ は算術右シフトとする．

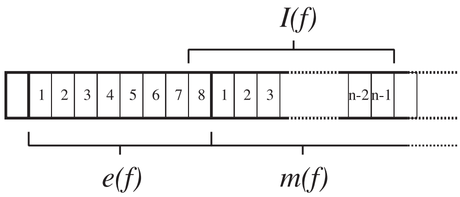

図 4 IEEE754 形式における浮動小数の形式
Fig. 4 Float format in IEEE 754.

4. 数値実験

4.1 乱数を用いた平均二乗誤差の評価

1 対の三次元ベクトルを区間 $[-1,1]$ 上の乱数で生成し，グラム–シュミット直交化によって第一と第二基底を求め，両者を外積し第三基底を求める．通常方式による基底を真値として，提案方式による基底の平均二乗誤差 (MSE) を計算し，評価を行う．なお，関数テーブルの各々の値は，IEEE754 方式による単精度仮数部 23 [bit] で表し，テーブルサイズは

$$\{23 \times 2^n [\text{Byte}] \mid n=0,1,2,3\}$$

の 4 種類を用いた．また，実験環境として，Windows XP Professional SP2，Intel Core2 Duo 1.86 [GHz] × 2, 0.99 [GByte] RAM を用いた．

1000 組の乱数ベクトルによる結果を表 1 と図 5 に示す．第一と第二基底の MSE はほぼ同じで，第三基底の MSE が最も大きくなる．これは，第三基底が第一と第二基底の外積で計算されるため，二つのベクトルの誤差が影響するためである．テーブルサイズが $n \geq 2$ の場合，いずれの基底も MSE が 10^{-4} 未満となるため，関数台 $[2^0,2^2)$ を 2^5 分割した，92 [Byte] の関数テーブルによって十分な近似精度が実現できる

表 1 関数テーブルサイズと平均二乗誤差
Table 1 Table size v.s. mean square error.

テーブルサイズ		平均二乗誤差		
n	[Byte]	第一基底	第二基底	第三基底
0	23	6.9×10^{-4}	7.2×10^{-4}	1.4×10^{-3}
1	46	1.7×10^{-4}	1.8×10^{-4}	3.5×10^{-4}
2	92	4.1×10^{-5}	4.4×10^{-5}	8.3×10^{-5}
3	184	1.1×10^{-5}	1.1×10^{-5}	2.3×10^{-5}

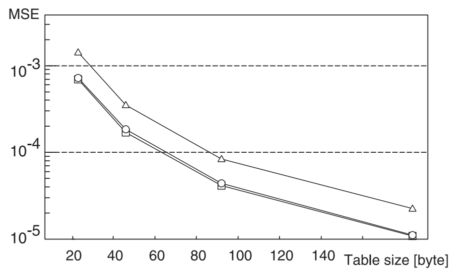

図 5 関数テーブルサイズと平均二乗誤差（横軸：テーブルサイズ [Byte]，縦軸：平均二乗誤差，節点：左から $n=0,1,2,3$），□：第一基底，○：第二基底，△：第三基底

Fig. 5 Table size v.s. mean square error. (Horizontal: table size[Byte], Vertical : avarage square error, nodes : from left, $n = 0,1,2,3$), □ : first base, ○ : second base, △ : third base

と考えられる．

4.2 ポリゴンデータを用いた速度・SNR の評価

局所直交変換を用いたポリゴンデータの復号処理を，通常方式と提案方式により実装し比較する．実験モデルとして，Stanford Computer Graphics Laboratory [9] において公開されている，“Dragon”，“Stanford Bunny”，“Armadillo” の三つを使用する．（表 2 参照）これらの位相情報は三角形リスト形式で記述されているため，TriStripper [10] を用い，一定の順路で頂点を重複なく巡る三角形ストリップ形式に変換する．

4.2.1 実験方法

文献 [4] の局所変換に基づいて実験を行う．まず，次の手順で符号化を行う．

（1） 三角形ストリップから p_n を読み込み，$\{p_{n-3}, p_{n-2}, p_{n-1}\}$ から座標系 (2) による局所変換を求める．このとき，通常方式または提案方式により正規化を行う．

（2） 差分ベクトル $d = p_n - \frac{p_{n-1}+p_{n-2}}{2}$ を変換する．

（3） 変換後の 3 成分を量子化ステップ 2^c により線形量子化する．

（4） 量子化後の 3 成分を符号部 1[bit]，指数部 8[bit]，仮数部 $(a-c)$[bit] で格納する．ただし，a は指数値を示す．

（5） 逆量子化を行い 3 成分を復号し，p_n を更新する．

（6） 頂点が終了するまで上記を反復する．

なお，$\{p_{n-3}, p_{n-2}, p_{n-1}\}$ が縮退している場合は，事前に計算された局所変換を用いる．

次に（4）の形式で格納された頂点情報を読み出し，上記の（1），(2)，(5）を反復することで三角形ストリップを復号する．

最終的に復号に要する時間及び復号結果を用いて比較を行う．復号後のポリゴンデータのひずみ [11] は，

$$\text{error} = \sum_{k=1}^{N} (V_k - V'_k)^2,$$

表 2 3 種類のモデルデータ
Table 2 Three polygon models.

モデル名	頂点数	ストリップ長
Dragon	433652	1284359
Stanford Bunny	34834	90972
Armadillo	172974	493400

$$\text{SNR} = 10 \log_{10} \frac{\sum_{k=1}^{N} (V_k - \text{center})^2}{\text{error}} \ [\text{dB}]$$

を用いて評価する．なお，N は頂点数，V_k は真の，V'_k は復号後の k 番目の頂点座標，center は全頂点の重心を示す．また，実験環境及び関数テーブルの仕様は **4.1** と同じとした．

4.2.2 評　　価

“Stanford Bunny” を用いてテーブルサイズを $n \in \{0,1,2,3\}$ とした場合の SNR の変化を図 6 に，復号時間の変化を図 7 に示す．テーブルサイズが増加するほど，関数近似の精度が向上し，SNR も向上することが分かる．**4.1** における実験結果と同様に，ひずみの改善は，$n = 2$ でほぼ飽和するため，92 [Byte] の関数テーブルで十分な精度が得られることが分か

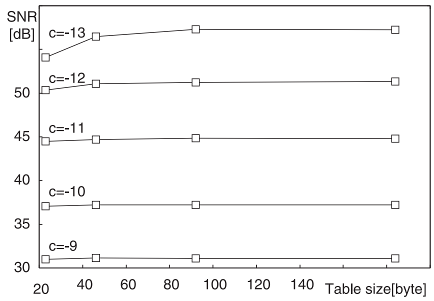

図 6 関数テーブルサイズと SNR（横軸：テーブルサイズ [Byte]，縦軸：SNR,[dB]，節点：左から $n = 0, 1, 2, 3$）
Fig. 6 Table size v.s. SNR characteristic curves. (Horizontal : table size [Byte], Vertical : SNR [dB], nodes : from left, $n = 0, 1, 2, 3$)

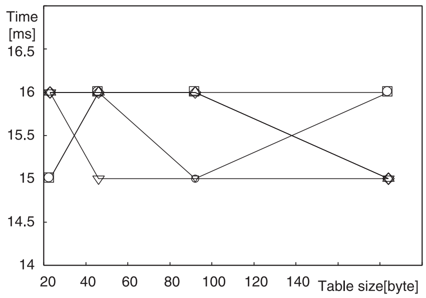

図 7 関数テーブルサイズと復号時間（横軸：テーブルサイズ [Byte]，縦軸：復号時間 [ms]，節点：左から $n = 0, 1, 2, 3$，□：$c = -13$，○：$c = -12$，△：$c = -11$，▽：$c = -10$，◇：$c = -9$）
Fig. 7 Table size v.s. decode time characteristic curves. (Horizontal : table size [Byte], Vertical : decode time [ms], nodes : from left, $n = 0, 1, 2, 3$, □ : $c = -13$, ○ : $c = -12$, △ : $c = -11$, ▽ : $c = -10$, ◇ : $c = -9$)

表 3 復号時間と SNR
Table 3 Decode time vs SNR.

Dragon

c	復号時間 [ms]		SNR[dB]	
	通常方式	提案方式	通常方式	提案方式
-13	406	234	55.8	54.8
-12	375	235	50.0	49.6
-11	375	234	43.8	43.6
-10	375	234	38.0	37.7
-9	375	250	32.1	31.2

Stanford Bunny

c	復号時間 [ms]		SNR[dB]	
	通常方式	提案方式	通常方式	提案方式
-13	31	16	57.4	57.3
-12	31	15	51.4	51.2
-11	31	16	44.9	44.9
-10	32	15	37.2	37.2
-9	31	16	31.0	31.1

Armadillo

c	復号時間 [ms]		SNR[dB]	
	通常方式	提案方式	通常方式	提案方式
-3	156	94	56.0	56.0
-2	157	94	48.8	48.8
-1	140	94	41.9	41.9
0	141	78	35.9	35.9
1	141	79	30.1	30.1

る．他方，復号時間はテーブルサイズに依存せず，15〜16 [ms] の間に収まる．なお，上記の二つの性質は，“Dragon” 及び “Armadillo” においても，同様に観察された．

最終的に，テーブルサイズを 92 [Byte] に固定した場合の復号時間と SNR を表 3 に示す．復号時間を見ると，提案方式は通常方式に比べ 50〜65%程度に短縮できている．SNR も，“Dragon” の $c = -13$ においては 1.0 [dB] の劣化が見られるものの，その他の誤差はわずかであり，また，“Stanford Bunny” と “Armadillo” においては通常方式とほぼ同等の SNR が得られていることが分かる．

5. む す び

本論文は，ポリゴンモデル圧縮手法の一つである局所変換において，ノルム二乗値の仮数部に対する小規模な関数テーブルを用いることで，正規化処理から平方根と除算を排除し，復号を高速化する方式を提案した．また，数値実験により，少ない SNR の劣化のもとで復号時間を 50〜65%に短縮できることを報告した．なお，ハードウェア実装を前提とした実機実装は今後の課題とする．

文 献

[1] C. Tauma and C. Gotsman, “Triangle mesh compression,” Graphics Interface ’98 Conference Proceedings, pp.26–34, 1998.

[2] M. Isenburg and P. Alliez, “Compressing polygon mesh geometry with parallelogram prediction,” Proc. Visualization 2002, pp.141–146, 2002.

[3] E. Lee and H. Ko, “Vertex data compression for triangular Meshes,” Pacific Graphics ’00 Conference Proceedings, pp.383–392, 2002.

[4] S. Gumhold and R. Amjoun, “High order prediction for geometry compression,” International Conference on Shape Modelling and Applications, pp.59–66, 2003.

[5] M. Deering, “Geometry compression,” SIGGRAPH 95, pp.13–22, 1995.

[6] G. Taubin and J. Rossignac, “Geometric compression through topological surgery,” ACM Trans. Graphics, vol.17, no.2, pp.84–115, 1998.

[7] M. Isenburg, P. Lindstrom, and J. Snoeyink, “Lossless compression of predicted floating-point geometry,” Comput. Aided Des., vol.37, no.8, pp.868–877, 2005.

[8] http://download.intel.com/jp/developer/jpdoc/ia32.pdf

[9] Stanford Computer Graphics Laboratory, http://graphics.stanford.edu/

[10] GPSnoopy’s Development Arena, http://users.pandora.be/tfautre/softdev/tristripper

[11] J. Li and C.C. Kuo, “Progressive coding of 3d graphics models,” Proc. IEEE, vol.86, no.6, pp.1052–1063, 1998.

（平成 21 年 7 月 16 日受付，10 月 4 日再受付）