GPUを応用したメッシュパラメタライゼーションの効率化

北海道大学　　○市川　大，　伊達　宏昭，　金井　理，　小野里　雅彦

要　旨

メッシュモデルのパラメタライゼーションは，テクスチャマッピングやリメッシングなどに応用されている．本研究では，共役勾配法を用いた，メッシュの稜線長と面分面積に関するひずみエネルギ最小化に基づくパラメタライゼーションの構築を，GPUを用いて効率化する手法を提案する．

1　はじめに

三角形メッシュモデルのパラメタライゼーションは図1のように，三角形メッシュモデルMと1対1対応する平面三角化グラフG（図1）を意味し，メッシュモデルのリメッシング・テクスチャマッピング・メッシュモデル間の合成などに応用されている[1]．しかし，このパラメタライゼーションの構築には，最適化問題や連立方程式を解くことが必要であり，変数の次元がメッシュ頂点数に比例し，大規模メッシュに対しては処理時間を要してしまう問題がある．

一方，近年のGPU（Graphics Processing Unit）は処理性能が向上し続けており，本来の目的であるグラフィックス専用の処理だけでなく，サウンド／ビデオエフェクト処理・流体解析シミュレーション[2]・形状スムージングの効率化[3]などの応用研究がなされている．

そこで本研究では，GPUを応用した3次元メッシュモデルのパラメタライゼーション構築の効率化手法を提案する．なお，本研究ではパラメタライゼーション構築手法として，三角形メッシュと平面三角化グラフ間の距離ひずみ・面分面積ひずみの最小化に基づいて行うMaillotらの手法[1]を効率化の対象とする．

2　パラメタライゼーション手法

Maillotらのパラメタライゼーション構築手法[1]は，式(1)の距離ひずみエネルギE_lと，式(2)の面分面積ひずみエネルギE_sとの重みつき線形和Eを最小化することで，メッシュ頂点iの座標$\mathbf{p}_i$に対応するパラメータ$\phi(\mathbf{p}_i)=\mathbf{q}_i=(u_i,v_i)$を求める手法である．

$$E_l=\sum_{i,j}\frac{\left\{\left\|\left(\phi(\mathbf{p}_i)-\phi(\mathbf{p}_j)\right)\right\|^2-\left\|\mathbf{p}_i-\mathbf{p}_j\right\|^2\right\}^2}{\left\|\mathbf{p}_i-\mathbf{p}_j\right\|^2} \tag{1}$$

$$E_s=\sum_{i,j,k}\frac{\left\{\det\left[\left(\phi(\mathbf{p}_j)-\phi(\mathbf{p}_i)\right)^T\left|\left(\phi(\mathbf{p}_k)-\phi(\mathbf{p}_i)\right)\right.\right]/2-S_{ijk}\right\}^2}{S_{ijk}} \tag{2}$$

ここで，S_{ijk}は三角形メッシュモデル内の3つの頂点i，j，kで構成される三角形の面積である．

Maillotらの手法では，このエネルギを最小化するために共役勾配法が用いられ，本研究では，この共役勾配法の演算をGPU上に実装し，処理速度の高速化を図ることを目的とする．以下にその手法を述べる．

3　GPUを応用した共役勾配法の効率化[3]

3.1　共役勾配法

共役勾配法は，制約のない最小化問題に対する解法の1つで，目的関数Eを最小化する制御変数ベクトル$\mathbf{x}$を，目的関数Eの勾配ベクトル∇Eを用いながら反復的に探索する方法である．

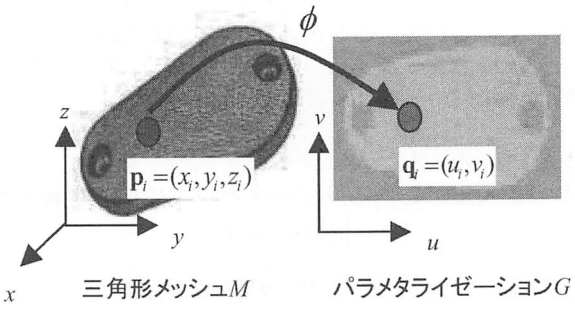

図1　パラメタライゼーション

制御変数ベクトル$\mathbf{x}$は更新式$\mathbf{x}^{(k+1)}=\mathbf{x}^{(k)}+\alpha^{(k)}\mathbf{p}^{(k)}$により更新される．$\mathbf{p}^{(k)}$は探索方向ベクトルと呼ばれ，式(3)で定義される．

$$\mathbf{p}^{(k+1)}=-\nabla E^{(k)}+\frac{\left\|\nabla E^{(k)}\right\|}{\left\|\nabla E^{(k-1)}\right\|}\cdot\mathbf{p}^{(k)} \tag{3}$$

また，探索ステップ幅$\alpha^{(k)}$は，式(4)で定められる．

$$\alpha^{(k)}=\frac{-\mathbf{p}^{(k)T}\nabla E^{(k)}}{\mathbf{p}^{(k)T}\mathbf{A}^{(k)}\mathbf{p}^{(k)}} \tag{4}$$

ここで，$\mathbf{A}^{(k)}$は目的関数$E^{(k)}$のヘッセ行列である．共役勾配法では，式(3)，(4)の中に現れる行列とベクトルの積演算，およびベクトルの内積演算が反復されるため，これらの演算をGPU上でテクスチャ間演算として計算することで効率化を図る．以下にその手法を述べる．

3.2　ランダム疎行列とベクトルの積

目的関数を2節で述べたEとした場合，そのヘッセ行列$\mathbf{A}$は疎行列となる．そこで，疎行列$\mathbf{A}=\{a_{\alpha\beta}\}$とベクトル$\mathbf{z}=\{z_\beta\}$の積（$\mathbf{y}=\mathbf{Ax}$）を効率よく計算するために，図2のように，GPUのテクスチャメモリ上の配列へ，$\mathbf{A}$および$\mathbf{x}$を以下の手法で配置し，演算を行う．

まず，疎行列$\mathbf{A}$の各成分を対角成分と非ゼロ非対角成分に分けて，それぞれテクスチャメモリ上の2次元テクスチャ配列A_i^x，A_j^aに格納する．A_j^aには，非ゼロ非対角成分数の少ない行の順に，A_j^aの2次元配列の上の行から行ごとにその非ゼロ成分を格納する．また，ベクトル$\mathbf{x}$は$\mathbf{A}$の対角成分と同じ形式で，別のテクスチャ配列X^xに格納する．さらに，非ゼロ非対角成分$a_{\alpha\beta}$との積が計算されるベクトル$\mathbf{x}$の成分x_βのX^x上のアドレスを，2次元テクスチャ配列C^aに格納する．また，A_j^aに格納された成分で行α中に最初に出現する成分の配列要素のアドレスをテクスチャ配列R^xの要素に格納する．

$\mathbf{y}=\mathbf{Ax}$の演算は，Nを$\mathbf{A}$の行数とすると$y_i=a_{ii}x_i+\sum_{j\neq i}a_{ij}x_j$，$(i=1,..,N)$と書ける．このうち，右辺第

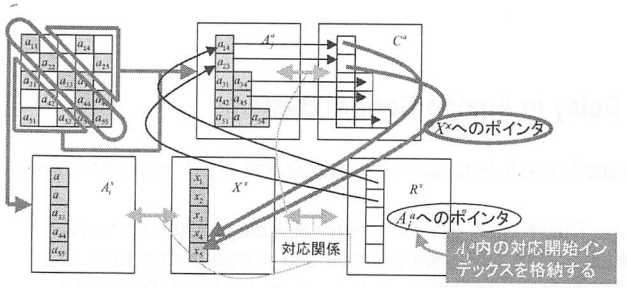

図2　疎行列とベクトルのテクスチャ上への格納方法

1項の成分同士の積演算 $a_{ii}x_i\ (i=1,\ldots,N)$は，GPU上で$N$個分をテクスチャ演算で一括して計算できる．一方，第2項は同数の非ゼロ非対角成分を持つ行ごとに，成分間の積とそれらの和を一括して計算する．これにより，疎行列とベクトルの積が効率化される．

3.3　帯状疎行列とベクトルの積

一般的な疎行列とベクトルの積演算の効率化は 3.2 節の方法で行えるが，帯状の疎行列の場合，より単純にGPUを用いて効率化できる．この場合，図3に示すように，非ゼロの帯1本ごとにひとつの2次元テクスチャ配列に格納する．積をとるベクトルも同様に 2 次元テクスチャ配列に格納する．非ゼロの帯1本が格納されたテクスチャとベクトルの要素が格納されたテクスチャ同士の積を求め，それぞれの積を帯の数だけ加算する．これによって帯状の疎行列とベクトルの積演算をGPU上で実装することが可能となる．

3.4　ベクトルの内積演算

$\|\nabla E\|$の内積計算を，GPU上で効率よく計算する方法は以下のとおりである．まず，2つのベクトル成分ごとの積を，3.2節と同様の方法で一括して計算し，それらの値をベクトルと同じ要素数の2次元テクスチャ配列の各要素に書き込む．次に，そのテクスチャ配列の要素の総和を，以下のSUMリダクションを用いて計算する．

通常，サイズが$N\times N$のテクスチャ配列に格納されているN^2次元ベクトル間の内積の和部分を，順次に加算していくとN^2回の計算パスが必要となる．SUMリダクションでは和の効率化を図るために，図4のようにテクスチャ配列上の2×2の隣接要素ごとに加算演算を行い，その和の値をサイズが$N/2\times N/2$のテクスチャ配列の対応した要素に書き込む．この操作を，最終的にテクスチャサイズが1×1になるまで繰り返すことで，全ての要素の総和が最終のテクスチャメモリ上に書き込まれる．この処理は，テクスチャ画像のミップマッピングと同様の処理で並列的に行えるため，結果として，$\log_2 N$回の計算パスですむこととなる．

4　パラメタライゼーション構築のGPU上での実現

2節で述べたパラメタライゼーション構築手法におけるヘッセ行列$\mathbf{A}$の成分は，Eのuとvの2階の導関数となる．この導関数は，式(1)(2)より，ある頂点に対しては隣接する頂点に対応する項のみが非ゼロとなり，その他はゼロとなる．よって，行列$\mathbf{A}$は疎行列となる．頂点の価数が不規則なイレギュラーメッシュは，行列$\mathbf{A}$がランダム疎行列になるので3.2節の手法を適用できる．一方，頂点の価数が特異点を除いた全ての頂点で6のセミレギュラーメッシュは，行列$\mathbf{A}$が帯状疎行列になるので，3.3節の手法が適用できる．入力メッシュの性質に合わせて，それぞれの手法を使い分けることで，パラメタライゼーション構築を効率よく実現できる．以上より，2節のパラメタライゼーション構築のための最小化問題を共役勾配法で解く際，3.2，3.3節で述べた疎行列とベクトルの積に対するGPUを用いた演算を利用した実装が可能となる．

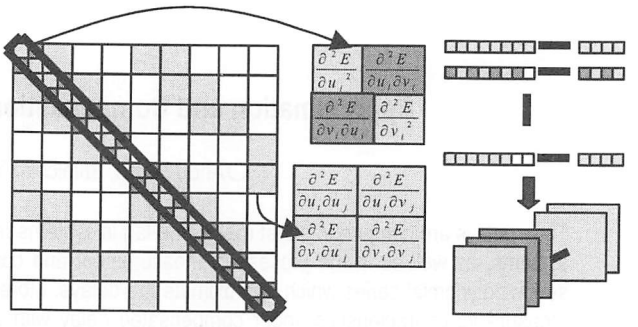

図3　帯状疎行列の非ゼロ要素のテクスチャ格納方法

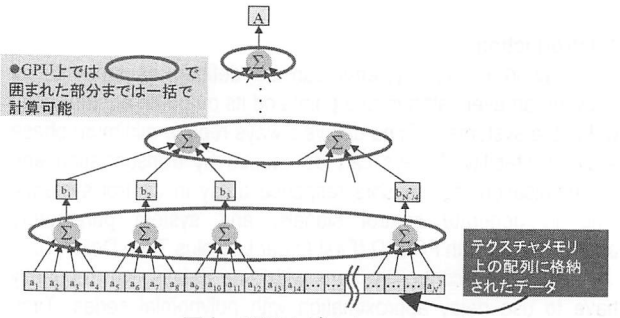

図4　SUMリダクション

表1　帯状疎行列とベクトルの積演算の結果

行列のサイズ	時間(GPU)	時間(CPU)
16284×16284	15〜32 ms	6.0〜6.3 s
4096×4096	15〜32 ms	141〜359 ms
1024×1024	15〜31 ms	0〜16 ms
256×256	15 ms	0〜16 ms

5　基礎実験とまとめ

非ゼロの帯が 5 本の帯状疎行列とベクトルの積演算にかかる時間(行列の生成時間は除く)を，GPU(GeForce 6600)を用いた場合とCPU(Xeon 2.8GHz)を用いた場合について比較した結果を表1に示す．GPUを用いた場合，一括演算できるテクスチャのサイズは一定であるので，そのサイズ内の疎行列サイズの変化と処理時間はほぼ相関が無く，GPUを用いない場合と比べ，演算の効率化が出来ていることがわかる．

本報告では，三角形メッシュモデルのひずみエネルギ最小化に基づくパラメタライゼーションの構築を，GPUを用いた共役勾配法に帰着させ，効率化する手法を提案し，基礎実験においてGPUにより疎行列とベクトルの積演算が効率化されることを確認した．今後の課題として，本手法のGPU上でのパラメタライゼーション構築の実装を済ませ，処理の効率化の効果を定量的に評価することがあげられる．

[参考文献]

[1] J. Maillot, H. Yahia, and A. Verroust, “Interactive Texture Mapping,” *Proc SIGGRAPH '93*, pp. 27-34, 1993.

[2] Randima Fernando 編，中本　浩 監: GPU Gems2 日本語版．ボーンデジタル社. 2005.

[3] J. Bolz et.al., “Sparse Matrix Solvers on the GPU : Conjugate Gradients and Multigrid,”, *Proc. SIGGRAPH* ‘03, pp.917-924, 2003.