グラフィクスとCAD 87－8
（1997. 10. 17）

# グラフィックス・アクセラレータを利用した3次元LIC法の高速化

喜久川誠, 藤田 昇, 茅 暁陽, 今宮淳美

山梨大学工学部電子情報工学科

我々は先行研究で, 従来2次元の構造格子だけに適用が制限されていたLIC法を, 3次元の任意面上の流れを可視化できるように拡張した. 入力画像にソリッドノイズを用い, 流線計算を3次元空間内で直接行なうことで, どのような形状の物体表面であっても歪みのないテクスチャが生成でき, 正確な流れの可視化が可能である. 本論文では, より良好な対話性を確保する目的で, この3次元LIC法の高速化を試みる. 計算時間の半分以上を占める視線と物体との交差判定部分をグラフィックス・ハードウェアに任せることにより高速化は実現される. 実装した方法を具体的な流体の可視化問題に適用し, その効果を検証する.

# Graphics Hardware Assisted Acceleration of 3D LIC

Makoto Kikugawa, Noboru Fujita, Xiaoyang Mao and Atsumi Imamiya

Department of Electrical Engineering and Computer Science Yamanashi University

*Line Integral Convolution(LIC) is a powerful texture-based vector field visualization technique. By convolving white noise input images with the filter kernels defined along the local vector directions, LIC can generate texture images effectively revealing the global and complex structures of a flow field. In our previous research, we have successed in extending the conventional LIC algorithm, whose use has been restricted only to 2D/3D structured grids, for visualizing the flows over arbitrary 3D surfaces. This paper presents a new technique for accelerating the proposed 3D LIC algorithm through utilizing the Z-buffer rendering hardware of modern graphics computer.*

## 1 はじめに

流れ場の可視化における有力な手法の一つに, 線積分畳み込み法 (Line Integral Convolution : LIC) がある [1]. これは入力画像であるホワイトノイズを流れの方向にぼやけさせることで, 流れを表すテクスチャを生成する手法である. 先行研究において著者らは, 従来2次元や3次元の構造格子だけにその適用が制限されていたLIC法を, 3次元の任意面上の流れを可視化できるように拡張した [3]. 同法では, ソリッドテクスチャリング技法を利用し, 3次元物理空間内で直接流線計算を行なっている. これにより, 物体の形状を考慮することなく, 歪みのないテクスチャを生成することができる.

本論文では, 我々が提案したこの3次元LIC法の高速化を試みる. 従来は, 同法の行程の一つであるピクセルごとの視線と物体との交点判定をソフトウェアで行なっていたために, 1枚の画像を生成す

るのに多くの時間を要していた．そこで，その交点判定部分をハードウェアで処理させることにより，処理時間の大幅な短縮が期待できる．

まず次節において，我々が提案した3次元LIC法の手順を紹介し，続く3節で本論文で提案する高速化の原理を説明する．4節で実装した本法を，具体的な流れの可視化問題に適用して前手法と比較し，その高速化の効果を検証する．最後に5節で将来の研究への指針を述べて，本論文をまとめる．

# 2 3次元LIC法

## 2.1 LIC法

LIC法は，1993年にCabralとLeedomにより提案されたベクトルフィールドの可視化技法である[1]．図1にその概念図を示す．規則的な直交格子上に定義されている2次元のベクトルフィールドが与えられたとする．LIC法では，それと同じ大きさのホワイトノイズ画像を入力し，それをベクトルフィールドの局所的な流線上に定義される1次元フィルタカーネルと畳み込みすることにより，出力画像を生成する．

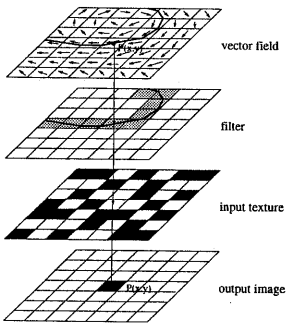


図1: ＬＩＣ法

LIC法は，流れの微細な構造や大域的な特徴を直観的かつ効果的に可視化することができる．その有用性により現在もたいへん注目されており，さまざまな拡張が行なわれている[4]．しかしながら，ベクトルフィールドと入力画像の1対1対応が必要であるなどの制限から，従来は2次元データにその適用が制限されていた．テクスチャマッピングを用いることで3次元の構造格子に拡張したLIC法も既に提案されている[2]が，マッピングによりテクスチャが歪んでしまうなどの問題点がある．

## 2.2 ソリッドテクスチャリングを用いた3次元LIC法

そこで我々は，ソリッドテクスチャリング[5]を用いることにより，LIC法を任意の3次元面上の流れを歪みなく可視化できるように拡張した[3]．図2にその提案手法の原理を示す．まず対象となる3次元任意面に向かって，視点からスクリーンの各ピクセルを通る視線と面との交点を求める．そして視点から見える交点だけに対して局所的な流線を計算し，3次元ホワイトノイズ画像との畳み込みを行なってそのピクセルの色を決定する．

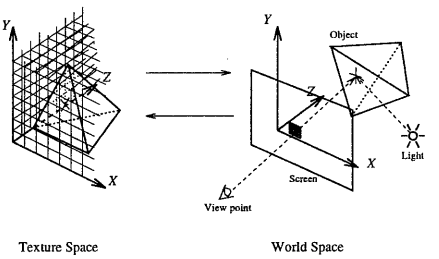


図2: 3次元ＬＩＣ法

### 2.2.1 ソリッドホワイトノイズ

3次元空間のある点$P(x,y,z)$におけるソリッドホワイトノイズの値は，文献[5]で提案される手法を使って手続き的に計算される．$x,y,z$座標がすべて整数値をとるような点により構成される3次元格子を想定する．各格子点でのノイズ値$V(i,j,k)$は，座標値$i,j,k$を引数とする疑似乱数へのハッシュ関数によって決められる．

```
int Noise(int i,j,k)
{
  int n = INDEX[(a*i) & 0x000000ff]
        + INDEX[(b*j) & 0x000000ff]
        + INDEX[(c*k) & 0x000000ff];
  return(WHITE_NOISE[n % TABLEN]);
}
```

ここで, WHITE_NOISE は二値のノイズがランダムに格納されているテーブルであり, TABLEN はそのテーブルの大きさである. INDEX は WHITE_NOISE テーブルを参照するための間接的なテーブルであり, 疑似乱数による整数値が格納されている. 対象となる 3 次元任意面をこの整数格子の中に置かれていると想定すると, 面上の任意の点 $P(x,y,z)$ におけるノイズ値は単純に各座標の floor をとり, その値をハッシュ関数の引数とすることで決定できる.

$$V(x,y,z) = Noise(\lfloor x \rfloor, \lfloor y \rfloor, \lfloor z \rfloor)$$

ある点でのノイズの値はその点の座標値に依存するので, 面の位置や向きにより各点でのノイズの値が変わってしまうことを防ぐために, ここでは座標系を, ソリッドノイズを合成するためのもの (テクスチャ空間) と, レンダリングのためのもの (ワールド空間) の 2 つに分けている. ワールド空間での視線とオブジェクトとの交点は, ノイズ関数を呼び出す前にテクスチャ空間上の点に変換される. 次項で説明するように, 局所的な流線の決定および畳み込み計算も, そのテクスチャ空間内で行なう.

### 2.2.2 3 次元物理空間内での LIC 計算

我々が提案した 3 次元 LIC 法では, 入力データとして汎用性の高い三角形メッシュを仮定している. そして三角形の各頂点に, 頂点座標とともにその点での流れのベクトルデータが格納されている. 3 次元物理空間内でソリッドホワイトノイズとの畳み込みを行なうためには, 視線と物体との交点からの流線を計算する必要があるが, これは三角形の頂点に定義されているベクトルを補間することで求めている. 流線計算を行なうには, まずその交点の落ちている三角形を特定する必要がある. また, ソリッドノイズの値を特定するためには, 上述のようにその交点の座標値を特定しなければならない. 提案した手法では, 各ピクセルごとに全三角形を走査し, 視線が落ちる三角形を特定し, さらにその交点座標を求めてその値をローカル座標系に返し, 特定した三角形および交点座標を使って流線の計算を行なっている. 以上のようなアルゴリズムにより, 物体の形状を考慮することなく, またテクスチャの歪みを起こすことなく, 3 次元の任意面上の流れ場の可視化を可能にした.

### 2.2.3 適用例

このソリッドテクスチャリングを用いた 3 次元 LIC 法を, 汎用ビジュアリゼーションソフト AVS [1] 上で実装し, 具体的なデータに適用した結果を図 3 に示す. ここで入力データは, 超音速場における space plane の機体表面上の空気の流れを数値シミュレーションした結果である. 元のデータは機体表面を最下部とした 3 次元の構造格子で表されていて, その中の表面から一格子分上の面での流れ場に注目し, 元の四角形格子を対角線で分割することで三角形メッシュのデータ形式に変換し, 3 次元 LIC 法を適用している.

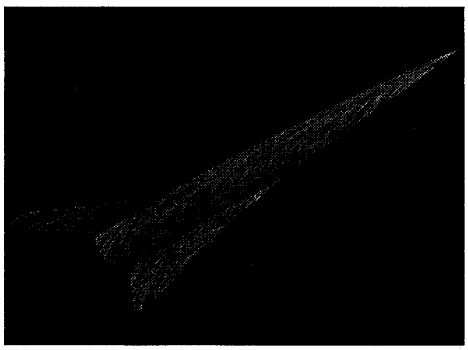

図 3: 3 次元 LIC 法を適用した画像

## 3 グラフィックスハードウェアを用いた高速化

流れの可視化においては, ユーザがさまざまな角度から物体表面を観察したりするなど, 対話的な操作が必要となる. このためには, ビューの再設定に対して高速に再計算が行なわれなければならない. 我々の提案した手法では, ビューを変えるたびにレイトレースをやり直さなければならず, 従来のままでは処理時間の点において問題がある.

[1] AVS は Advanced Visual Systems Inc. の商標である.

3 次元 LIC 法には,

1. 三角形の特定
2. 交点座標の特定
3. LIC 計算

の三つの行程がある. 本論文では, ハードウェア Z バッファリング機能を利用してこのうち行程 1 と 2 の高速化をはかる. 現在処理時間中の 6 割〜7 割を占めるこの 2 つの行程が高速化できれば, 結果的に全体の処理時間を大幅に短縮できると考えられる.

ハードウェア上で Z バッファ処理を行なった場合, スクリーンにもっとも近い Z 値が Z バッファに保存される. 3 次元 LIC 法で必要となる交点とは, 視線が最初に物体と交差する点である. すなわち, Z バッファの値を利用すれば交点座標を高速に求めることが出来る. また, 三角形の特定にはフレームバッファを利用する. 物体を構成する三角形ごとに違う色を割り当てておくことで, フレームバッファに保存された値から, あるピクセルを通る視線がどの三角形に落ちているのかを求めることが可能である.

提案する手法のアルゴリズムを以下に示す.

```
 0 n = 全三角形数;
 1 for(i=0;i<n;i++) {
 2  triangle_color(color[i]);
 3  define_triangle(vertex1,vertex2,vertex3);
 4 }
 5
 6 Z-buffering();
 7
 8 n = 画像幅 * 画像高さ;
 9 for(i=0;i<n;i++) {
10   pixel_color = frame_buffer[i];
11   triangle_index = index_search(pixel_color);
12   Z_coordinate_value = Z_buffer[i];
13   world_xyz = z-to-world_xyz(Z_coordinate_value);
14   local_xyz = world-to-local(world_xyz);
15   pixel[i]
16    = LIC(triangle_index, local_xyz);
17 }
```

1〜4 行目では, ハードウェアレンダリングを行なうために, 形状データを処理系に渡す. 2 行目では, 三角形ごとに異なる色を割り当てる. そしてハードウェア Z バッファリング処理を行なった後, 各ピクセルごとにフレームバッファ, Z バッファからそれぞれ値を取り出し, 三角形と交点座標を特定する. 11 行目では, 2 行目で割り当てた色から逆引きをすることで三角形を決定する. 13 行目では, Z バッファから取り出した値とピクセル座標, 視点座標とから, x, y 座標を求めることにより交点座標値を算出する. こうして求めた三角形と交点座標から, 従来通り LIC を計算して, 対応ピクセルの最終的な値を決定する.

## 4 実験結果

前節で述べたアルゴリズムを, AVS と Open GL を用いて SGI のグラフィックスワークステーション Indy (Iris 5.3 OS, R5000 150MHZ CPU, メモリ 64MB) 上で実装し, 具体的なデータに適用して以前の手法と比較してどれだけ高速化がなされるかを検証した.

今回は図 3でも示した space plane のデータを用いて, 入力する三角形メッシュの個数, および画像サイズを変化させ, 前手法と改良した手法のそれぞれの実行時間を測定した. 表 1にその結果をまとめる.

| (メッシュ数) | 1000 | | 10000 | |
|---|---|---|---|---|
| (画像サイズ) | 320×240 | 640×480 | 320×240 | 640×480 |
| 前手法 | 17 | 55 | 165 | 621 |
| 改良手法 | 5 | 5 | 20 | 20 |

表 1: 実行結果 (時間の単位は秒)

表 1では, メッシュの数を 1000 個および 10000 個, 画像サイズを 320×240 および 640×480 とし, その組合せによる 4 通りのケースを示している. まず単純に前手法と比較してみると, すべてのケースにおいて改良した方の処理が速くなっており, 改良手法の高速化が実証されている. 特に三角形メッシュの個数および画像サイズがともに大きい場合については, 劇的な高速化がなされている.

前手法では, 各ピクセルごとに物体を構成する全三角形を走査して視線との交点判定を行なっている. このため, 処理時間は画像の大きさだけでなく三角形の個数にも大きく依存しており, 特に三角形の数が 1 万個程度になると, 表に示される通り比較的小さなサイズの画像を表示するときにもかなりの時間を要している. これに対し, 今回提案した高速化手法では, 三角形の特定, 交点座標の算出ともに, 簡単

な式で一時に求めることができ, 繰り返し処理を全く必要としない. 表からもわかるように, 処理時間が三角形の個数に依存しない提案手法は, 探究すべきシーンの複雑さに依存しない一定のフレームレートが保証される, 良好な対話的可視化環境を提供できる.

## 5 まとめと今後の課題

本研究では, 我々が提案した3次元LIC法を, グラフィックスハードウェアを用いることで高速に処理できるように改良した. 処理時間の半分以上を占めていたピクセルごとの三角形の特定と交点座標の算出を, ハードウェアZバッファリングを利用することにより, 物体表面上の三角形の個数に依存することなく高速に行なえるようにした. さらに具体的なデータに適用し, 同法の高速性を実証した. また同法は標準的なグラフィックスライブラリであるOpenGLを用いることで汎用性に富んでいる.

今後の課題としては, 処理のもう一つの行程である3次元物理空間でのLIC計算の高速化があげられる. 2次元LIC法については既に高速アルゴリズムが提案されており [6], それを3次元LIC法に適用することなどが考えられる.

## 謝辞

本研究を進めるにあたって多くの御助力をいただいた (株) 日本シリコングラフィックスの寺田 健彦さんに感謝します. また, space planeのデータを御提供下さった宇宙科学研究所の藤井孝蔵助教授に感謝します.

## 参考文献

[1] B. Cabral and C. Leedom. (1993). "Imaging Vector Field Using Line Integral Convolution." *Proceedings of SIGGRAPH'93*, ACM SIGGRAPH, pp. 263-270.

[2] L.K. Forssell and S.D. Cohen. (1995). "Using Line Integral Convolution for Flow Visualization: Curvilinear grids, Variable-speed Animation and Unsteady Flows" *IEEE Transaction on Visualization and Computer Graphics*, Vol.1, No.2, pp. 133-141.

[3] X. Mao, M. Kikukawa, N. Fjita and A. Imamiya (1997) "Line Integral Convolution for Arbitary 3D Surfaces through Solid Texturing." *Proceedings of the Eight Eurographics Workshop on Visualization in Scientific Computing*, EuroGraphics, pp. 67-76.

[4] "Texture Synthesis with Line Integral Convolution" *Course Notes for SIGGRAPH'97*, ACM SIGGRAPH.

[5] K. Perlin (1985) "An Image Synthesizer." *Computer Graphics*, Vol. 19, No.3, pp. 287-296.

[6] D. Stalling and H.-C. Hege. (1995) "Fast and Resolution Independent Line Integral Convolution." *Proceedings of SIGGRAPH'95*, ACM SIGGRAPH, pp. 249-256.