# 3.10 リアルタイム3D グラフィックスにおける プログラマブルシェーダ技術の概要

# Introduction to Programmable Shader in Real Time 3D Computer Graphics

**上村周平、桐井敬祐、松村誠、松本健一郎**

シリコンスタジオ株式会社 コンテンツデベロップメント事業部

〒150-0013 東京都渋谷区恵比寿1-21-3

**Syuhei UEMURA, Keisuke KIRII,**

**Makoto MATSUMURA, Kenichiro MATSUMOTO**

Contents Development Business, Silicon Studio Corporation

1-21-3 Ebisu Shibuya-ku, Tokyo 150-0013 Japan

Nevertheless the visualization of large-scale data had played the important role which influences informational usefulness in the basic field of science, the high-end graphics system or the exclusive system needed to be used. On the other hand, in recent years, the progress speed of the capability of the video game console or the graphics board for PC has a remarkable thing reflecting the expansion tendency of TV game market in and outside the country. Especially, the "programmable shader" technology in which the several graphics chip maker has started implementation is the innovative technology which can also be called change of generation of real-time 3D graphics, and the scope of the visual expression technique has spread greatly. However, it cannot say that the development/use environment of software which used programmable shader are fully generalized, and the present condition is that the grope of the applied technology to overly the ultra high-speed/quality visualization of large-scale data is not prograssing. We provide the outline of programmable shader technology and consider the possibility of the application to large-scale data visualization.

**Keywords : Programmable Graphics Hardware, Shading Language, Parallel Processing, Isosurface**

## 1. はじめに

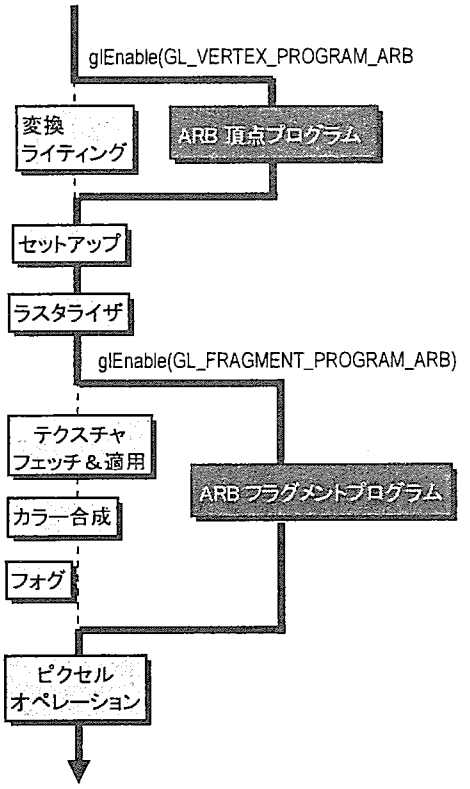
従来型のリアルタイム CG 用ハードウエアの描画パイプラインにおいては、各処理ユニットが特定の固定的な機能を持っており、描画処理内容の制御は処理モードの変更による方法のみが可能であった。プログラマブルシェーダとは、リアルタイム CG 描画パイプラインにおけるシェーディング処理部分をプログラム可能にするための技術である。本来シェーディング処理はポリゴンの塗りつぶし(陰影付け)を意味するが、プログラマブルシェーダにおいては頂点属性(座標値、色、法線など)の制御も含んだ概念となっている。グラフィックスハードウエアに実装されたプログラマブルシェーダ機能にアクセスするには、現在のところ OpenGL もしくは Direct3D の2種類の API が存在するが、ここではプラットフォーム非依存の業界標準である OpenGL1.4 インターフェース[1]について概説する。

## 2. ARB 頂点プログラムの概要

頂点プログラムは、ユーザ(開発者)が独自にプログラム可能な変換/ライティングプロセッサ(T&L ユ

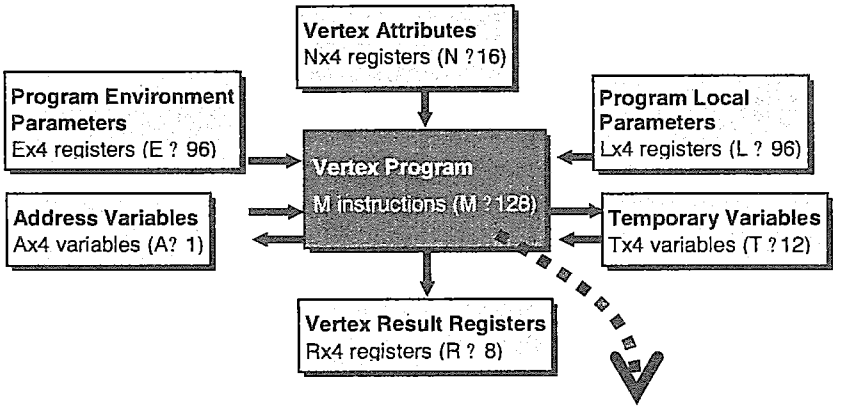
ニット)上で動作する。このような仕組みを利用する事により、プログラマによる頂点属性処理の総合的なコントロールが可能になった。これは、ユーザ(ソフトウエア開発者)が定義した複雑な頂点処理のハードウエア・アクセラレーションが可能である事を意味しており、従来は CPU にて処理していたキャラクターアニメーションにおけるスキニング・頂点ブレンド等のアルゴリズムや、カスタマイズしたテクスチャ座標計算、テクスチャ行列演算などをグラフィックス・ハードウエア上で処理する事ができる。頂点プログラムはアプリケーションの実行時に動的にグラフィックス・ハードウエアに読み込ませる事ができるが、従来型の固定機能の使用と頂点プログラムの使用はモードを切り替えることにより使い分ける事ができる(図1)。

図1
頂点プログラムとフラグメントプログラムのスコープ




プログラムの内容は T&L ユニット上で動作するアセンブリ言語命令セットであり、基本的に「入力レジスタの値を読み込み、それを加工して出力レジスタに書き込む」というものである(図2)。頂点プログラムに対して入力することができるのは任意の頂点属性であり、演算結果として出力するのはプログラムによって変換された頂点属性である。この出力属性には、同次クリッピング空間(homogeneous clip space)での位置、色(表面・裏面、プライマリ・セカンダリ)、フォグ座標、テクスチャ座標、点サイズ等が含まれる。プログラム内では、頂点の生成や削除を行う事はできず、入力された1つの頂点に対して必ず1つの頂点が出力される。また、エッジ、表面、隣接する頂点など、トポロジカルな情報を与える事はできない。

図2 頂点プログラムの概念図




## 3. ARB 頂点プログラムの作成方法

これらの頂点プログラム機能の実装は、初期段階ではグラフィックスチップメーカが独自に行っていた関係で OpenGL でのインターフェースも各社個別の拡張として実現されていた。しかし現在は OpenGL ARB により統一され、ARB_Vertex_Program という形で標準化されている。

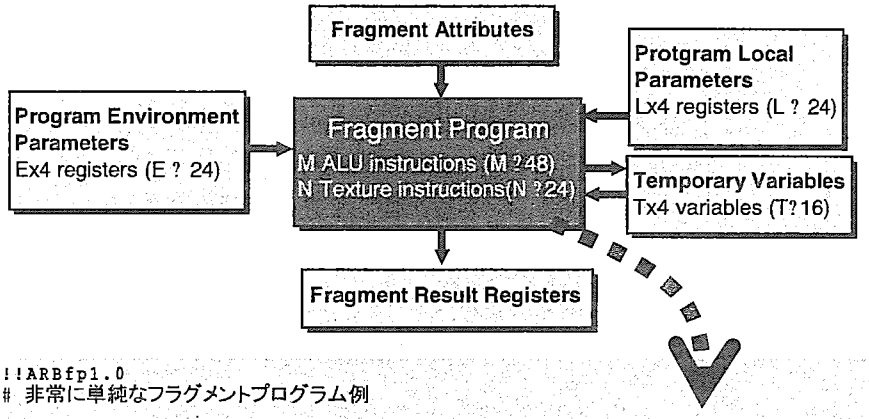
プログラムは OpenGL の Glubytes 型の配列(文字列)と

```
!!ARBvp1.0
# 非常に単純な頂点プログラム例

ATTRIB mypos = vertex.position;      # 頂点属性(頂点座標)のバインド
# カレントの OpenGL 状態(ModelViewProjection 行列)とバインド
PARAM mtx[4] = { state.matrix.mvp };

# 頂点座標の変換
DP4  result.position.x, mtx[0], mypos;
DP4  result.position.y, mtx[1], mypos;
DP4  result.position.z, mtx[2], mypos;
DP4  result.position.w, mtx[3], mypos;

MOV  result.color, vertex.color;     # プライマリ色をそのまま出力

END
```

して作成する。プログラムの作成と管理の方法はOpenGLの「テクスチャオブジェクト」と同様の考え方に基づいた、「プログラムオブジェクト」というメカニズムを用いる。プログラムで使用できるパラメータには頂点属性(Vertex Attributes)、ローカルパラメータ(Program Local Parameters)、環境パラメータ(Program Environment Parameters)のは3種類があり、各パラメータは、Begin/End ブロックの外で変更が可能である。

```
// 頂点プログラムの作成例
Gluint progid;

// プログラムオブジェクトのハンドルを生成する
glGenProgramsARB( 1, &progid );

// カレントのプログラムオブジェクトをバインド(作成)する
glBindProgramARB( GL_VERTEX_PROGRAM_ARB, progid );

// カレントのプログラムオブジェクトのプログラムをロードする
glProgramStringARB( GL_VERTEX_PROGRAM_ARB,
                    GL_PROGRAM_FORMAT_ASCII_ARB,
                    strlen(myString), myString );

.......

// プログラムオブジェクトを削除する
glDeleteProgramsARB( 1, &progid );
```

図3 フラグメントプログラムの概念図




## 4. ARB フラグメントプログラムの概要

従来型のパイプラインにおけるフラグメント毎処理はテクスチャの適用、カラー合成、フォグ処理などの限定的なものであった。幾つかの拡張機能でより高機能なテクスチャ環境やバンプマップに代表されるフラグメント毎ライティング処理が可能にはなったが、それらの機能も十分に柔軟性のあるものとは言えない。ARB フラグメント拡張は、アプリケーションによるフラグメント演算プログラム(命令シーケンス)の定義を可能にするメカニズムを提供するものである。プログラムモデルは頂点プログラムと同様のアセンブラベースのものでピクセル毎のSIMD 命令セットとなっている。フラグメントプログラムで使用するリソースは、フラグメント属性、テクスチャ、環境パラメータ、ローカルパラメータ、テンポラリ変数などである(図3)。フラグメントプログラムの生成・管理方法は、頂点プログラムと同様にプログラムオブジェクトの概念に基づいており、GL_FRAGMENT_PROGRAM_ARB というトークンを使用する。

```
!!ARBfp1.0
# 非常に単純なフラグメントプログラム例

TEMP tmp, tmp2;                          # テンポラリ変数
ATTRIB tex0 = fragment.texcoord[0]; # フラグメント属性(テクスチャ座標)のバインド
ATTRIB col0 = fragment.color;       # フラグメント属性(色)のバインド
PARAM mycol = { 0.9, 0.2, 0.2, 1.0 };# 定数パラメータ
OUTPUT out = result.color;               # 出力(リザルト)のバインド

TEX tmp, tex0, texture[0], 2D;           # テクスチャのフェッチ

MUL tmp2, col0, mycol;                   # フラグメント色と定数色をモジュレーション
MUL out, tmp2, tmp;                      # 上の結果とテクスチャのモジュレーション

END
```

## 5. シェーディング言語

標準化の遅延が原因で、チップメーカ各社が異なる方法で実装してきたプログラマブルシェーダのOpenGL 拡張機能は、上述の ARB 拡張の整備により、ようやく OpenGL の存在意義としての高度のポータビリティを実現できるレベルに到達しつつある。

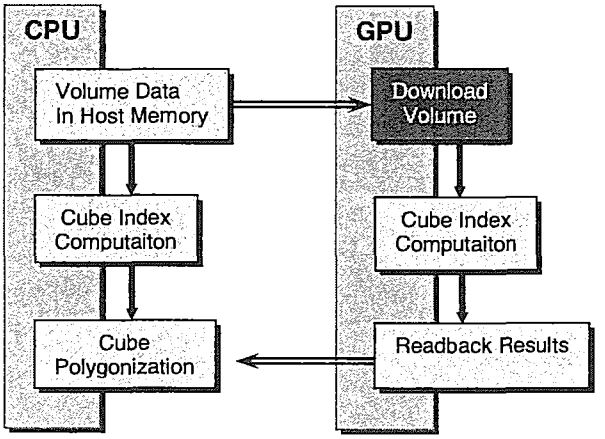
しかしながらシェーダプログラム自体はアセンブラ形式の低レベル言語を使用しなければならず依然として生産性向上の足かせとなっている。現在ソフトウエア開発で一般的に使用される言語は C 言語を代表

とする高級言語であるのと同様に、シェーダプログラム用の標準的高級言語仕様が望まれるのは自然の流れである。OpenGL ARB では、NVIDIA の Cg, Microsoft の HLSL 等各社が独自に開発を進めているシェーディング言語との互換性を考慮しながら業界標準としての OpenGL Shading Language(通称 Glslang)の仕様策定を進めている[2]。

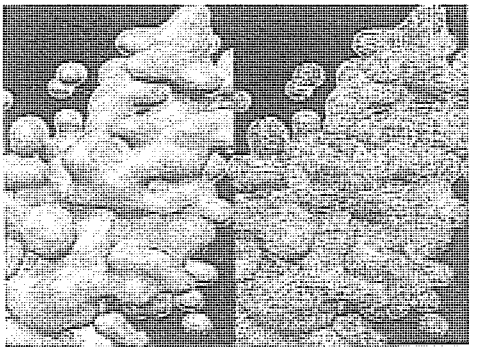
## 6. 応用技術

レイトレーシングやフォトンマッピングなど大域照明(グローバルイルミネーション)を実現するには至らないものの、プログラマブルシェーダ技術の登場により BRDF(双方向性反射率分布関数)やイメージベースドライティングなどローカルな照明モデルを自由にプログラミングできる事ができるようになるため、リアルタイムグラフィックスでの描画表現手法は新たな段階を迎えたと言える。製造業におけるデジタルプロトタイプやシミュレータ、エンターテイメント VR など対話性と写実性が求められる分野への応用は最も重要な課題であるし、今後フラグメントプログラムの精度やリソースが十分なレベルに到達すれば、リアルタイム映像に適用可能な高速画像処理プロセッサとしての利用法も考えられる。さらに、この高度な計算能力自体に注目する動きもある。従来から既存のハードウエアで高品質なシェーディングを実現するための研究は行われてきているがそれらの多くに共通する考え方はグラフィックスパイプラインを SIMD パラレルプロセッサとして利用しようとする点である。本来グラフィックスハードウエアはライティング・シェーディング機能を使ってグラフィックス描画を行う目的で設計されているわけであるが、この考え方に基づけば描画目的以外の用途にも応用する事ができるであろう。

図4 Accelerated isosurface polygonization
(前処理されたデータセットを適用する場合の処理の流れ)




シリコンスタジオではその1例として、マーチンキューブ・アルゴリズムによる等値面ポリゴン生成処理の一部を GPU で計算させる手法を考案した[3]。これによりメタボールや数値シミュレーション結果などの動的に生成されるボリュームデータをリアルタイムに描画することが可能となった。また、この手法はハードウエアによるピクセル処理結果を使用してポリゴンのジオメトリを構成するという、通常のグラフィックスパイプラインを逆行するアプローチである点でも非常にユニークであり、ピクセル処理ユニットからジオメトリ処理ユニットへデータを流すパスがハードウエア的に実装可能であれば、プログラマブルシェーダ応用の幅はさらに広がる可能性がある事を示唆している。



## 参考文献

[1]Mark Segal, Kurt Akeley, The OpenGL Graphics System: A Specification(Version 1.4)
[2]John Kessenich, Dave Baldwin, Randi Rost, The OpenGL Shading Language(Version 1.05)
[3] M.Matsumura, Silicon Studio Corp.(Japan); K.Anjyo, OLM Digital, Inc.(Japan) Accelerated isosurface polygonization for dynamic volume data using programmable graphics hardware., Visualization and Data Analysis 2003 conferences [5009-20]