GPU使って何しよう
～アーキテクチャから見たGPUの活用方法再考～

What should I do with the GPU
~ Reconsideration on how to utilize GPU as seen from the architecture ~

床井浩平[1)]
Kohe Tokoi

1) 和歌山大学システム工学部 (〒640-8510 和歌山県和歌山市栄谷930, E-mail: tokoi@sys.wakayama-u.ac.jp)

Application to general purpose computation (GPGPU, General Purpose GPU) of graphics display processor (GPU, Graphics Processing Unit) is mainly to utilize a large number of execution units called shader processors. In this presentation, we will also explain the idea of calculation method using graphics-specific units such as rasterizer and frame buffer.

Key Words : *GPU, GPGPU, Programmable Shader, Rasterizer, Frame Buffer*

1. はじめに

現在のコンピュータのグラフィックス表示を行うハードウェアに搭載されているプロセッサは GPU (Graphics Processing Unit) と呼ばれ，その高い演算能力からグラフィックス表示以外の用途にも用いられている．このような GPU の利用方法は GPGPU (General Purpose GPU) と呼ばれ，現在では機械学習や仮想通貨のマイニングなどの用途にも盛んに使用されている．

このような応用では，GPU に搭載されているシェーダプロセッサなどと呼ばれる多数の演算ユニットを活用した並列処理に重点が置かれることが多い．しかし，本来グラフックス処理用のハードウェアである GPU には，これ以外にラスタライザやフレームバッファ，テクスチャのサンプラなど，グラフィックス処理特有のハードウェアを備えている．これらを有効に活用すれば，特定の問題に対して，高い処理性能を得られる可能性がある．

本講演では GPU のこれらのグラフィックス処理特有のハードウェア機能の特徴を示し，それらの OpenGL[1] による活用例について紹介する．

2. GPUアーキテクチャの概念モデル

2.1. レンダリングパイプライン

コンピュータによる図形描画の方法は，レイトレーシング法などのサンプリングによる方法と，デプスバッファ法などのラスタライズによる方法の二つが主流である．このうちラスタライズによる方法では，図形のデータを入力したのちに描画命令（ドローコール）を発行することにより，ラスタ化処理により画像が生成される（図 1）．

GPU による図形描画（図 2）は，CPU から受け取った図形の頂点属性（位置，法線，色，テクスチャ座標など）やテクスチャ（画像）を一旦 GPU 上のメモリ（頂点バッファ）に格納し，点，線分，あるいは三角形からなる基本図形（レンダリングプリミティブ，図 3）を指定して描画命令を発行することによって行われる．

描画命令が発行されると，GPU は頂点バッファから頂点属性を取り出し，シェーダプロセッサ (Shader Processor, SP) の入力にセットして，それを起動する．シェーダプロセッサは入力された頂点属性に対して座標変換や陰影計算などを実行し，結果をラスタライザに出力する．この処理を担当するシェーダをバーテックスシェーダと呼ぶ．

ラスタライザはバーテックスシェーダから受け取った頂点属性をもとに塗りつぶし処理（走査変換，スキャンコンバージョン）を行い，図形をラスタ化する．その際，塗りつぶす画素における頂点属性の補間値を求め，それをシェーダプロセッサの入力に設定してシェーダプロセッサを起動する．

ラスタライザによって起動されたシェーダプロセッサは，入力された補間値をもとに画素単位の陰影計算やテクスチャのサンプリングなどを行なって最終的な画素の色を決定し，結果をフレームバッファに出力する．この処理を担当するシェーダをフラグメントシェーダと呼ぶ．

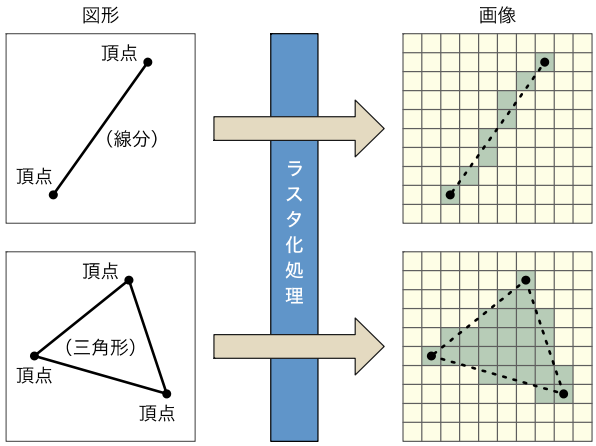

図 1 ラスタ化処理による図形描画

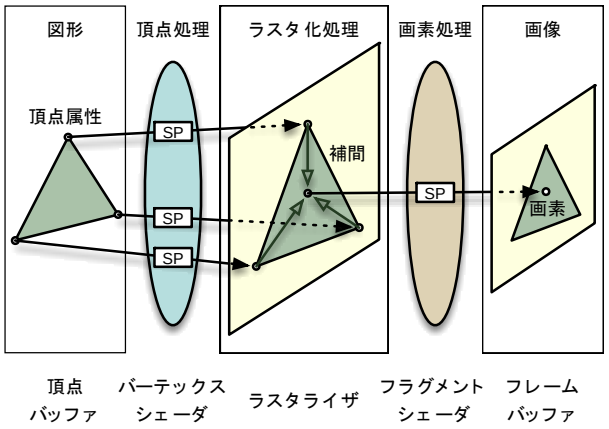

図 2 レンダリングパイプライン

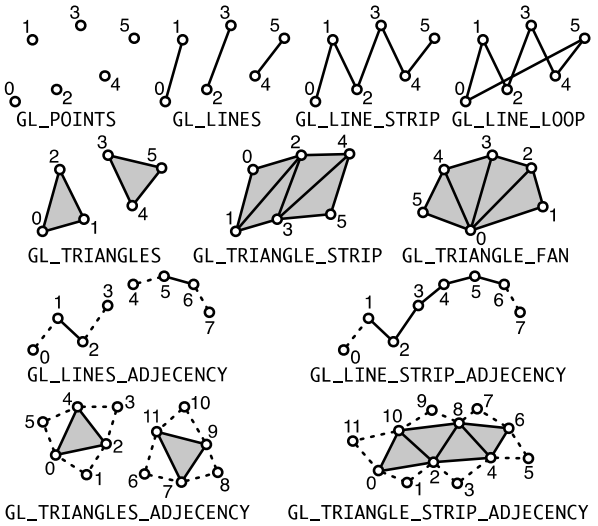

図 3 レンダリングプリミティブ

このように GPU による図形描画のプロセスは，処理が複数のステージに分割され，データを加工しながら順送りするパイプライン処理となっている．そのため，図形描画のためのこの機構，もしくはこの手順は，レンダリングパイプラインと呼ばれる．

2.2. GPU による計算モデル

GPU による計算処理の主体は，それに多数内蔵されているシェーダプロセッサである．GPU はこのシェーダプロセッサの並列処理により，高い計算能力を発揮する．

ただし，一個のシェーダプロセッサは一般的な CPU とは異なり，与えられた入力を処理して結果を出力するという一方向の処理しか行うことができない．したがって，これは与えられた入力データの集合から出力データの集合を生成する，写像のような処理になる．

シェーダプロセッサのこのような仕組みはレンダリングパイプラインを構成するのに適するが，比較的シンプルな構造のプロセッサを多数並列動作させることによる性能向上にも貢献している．

その一方で，出力したデータを（シェーダプロセッサ単体では）入力に書き戻すことができない，結果を出力した後はシェーダプロセッサによる処理を終了する必要がある（無限ループするとシェーダプロセッサは他のデータの処理に移れない），並列動作する他のプロセッサのデータを参照できない，などの制限がある．

そのため，処理が順序に依存するような従来型のプログラムを，そのまま GPU 上で高速動作させることは難しい．しかし，前述の性質を考慮して問題に適用すれば，その並列性を活かした高い処理性能を得ることができる．

2.2.1. 頂点処理

OpenGL では，GPU のメモリの管理にバッファオブジェクトという機構を用いる．CPU は最初に頂点属性を一旦このバッファオブジェクトに格納する．このバッファオブジェクトは頂点バッファオブジェクト (VBO)，あるいは単に頂点バッファと呼ばれる．

バーテックシェーダでは，シェーダプロセッサは頂点バッファから入力された頂点属性を処理して，その結果をレンダリングパイプラインの次のステージに出力する．図 2 に示した構成では次のステージはラスタライザだが，ジオメトリシェーダあるいはテッセレーション制御シェーダに入力して，図形のテッセレーション（細分化）を行うこともできる（図 4）．

バーテックスシェーダの出力は次のステージに送るだけでなく，別の頂点バッファオブジェクトに格納することもできる．この機能はトランスフォームフィードバックという．これにより，ここで互いに独立したデータの集合を別のデータの集合に変換する処理を，多数のシェーダプロセッサを用いて並列に行うことができる．

また，このときに次のステージでラスタ化やテッセレーションの処理を行わないようにすれば，GPU のシェーダプロセッサのみを利用した数値計算が可能になる．これは最もシンプルに GPU の数値計算能力を活用する手段として利用できる．

2.2.2. テッセレーション

テッセレーションは入力されたレンダリングプリミティブの細分化や別のレンダリングプリミティブへの置き換え，もしくは破棄などを行うことをいう（図 4）．OpenGL では，これを単独のジオメトリシェーダで行う場合と，テッセレーション制御シェーダとテッセレーション評価シェーダの組で行う方法がある．

テッセレーションの出力も，トランスフォームフィードバックによって別の頂点バッファオブジェクトに格納することができる．バーテックスシェーダの場合は入力された頂点属性と出力は一対一対応するが，テッセレーションの出力の場合はこの限りではない．

2.2.3. ラスタ化

ラスタライザはラスタ化の際に頂点属性の補間も行う．フラグメントシェーダのシェーダプロセッサはこの補間

図 4 テッセレーション

値を入力として用いる．したがってラスタライザは，多数のシェーダプロセッサの入力となる補間値を同時に生成する巨大な補間器として利用することもできる．

2.2.4. 画素処理

フラグメントシェーダでは，シェーダプロセッサはラスタライザによって割り当てられた画素における頂点属性の補間値を受け取り，それをもとにテクスチャなどを参照しながら，その画素における色を求めて深度値とともに出力する．この出力はフォグ処理やラスタオペレーション（アルファテスト，ステンシルテスト，デプステスト，ブレンド処理，ロジック処理）を経てフレームバッファに書き込まれる．また，画素の書き込みを行わずに処理結果を破棄することもできる．

フラグメントシェーダを利用して数値計算を行う場合は，たとえば画面のビューポート（表示領域）いっぱいのポリゴンを描けばよい．これにより，表示領域の画素ごとにシェーダプロセッサが起動され，結果がフレームバッファに格納される．このとき，結果の格納先としてユーザ定義のフレームバッファであるフレームバッファオブジェクト (Frame Buffer Object, FBO) を指定すれば，出力した内容をテクスチャとして参照することが可能になる．

2.2.5. ラスタオペレーション

フレームバッファへの書き込みの際に実行されるラスタオペレーションも，数値計算に活用できる．アルファテストやステンシルテストは格納先となる画素の書き込みマスクとして利用できるほか，デプステストは最小値／最大値選択，ブレンド処理は算術演算，ロジック処理は論理演算の機能である．

標準のフレームバッファに書き込まれた内容は画像化され，ディスプレイ上に表示される．一方，ユーザ定義のフレームバッファオブジェクトに書き込んだ内容は画面には直接表示されないが（Bit Blit により標準のフレームバッファに転送することは可能），書き込んだ内容はテクスチャとして参照できる．

フレームバッファオブジェクトの構成はプログラマブルであり，構成をユーザが設定できる．たとえば，標準のフレームバッファのカラーバッファが保持できる値は，画像表示に用いる [0, 1] の範囲の実数だが，フレームバッファオブジェクトでは，これに 16bit もしくは 32bit の整数や，16bit もしくは 32bit 浮動小数点の実数を指定することができる．

このとき，ブレンド処理として加算を指定すれば，フラグメントシェーダの出力結果がカラーバッファに積算される．すなわち，フレームバッファを巨大なアキュムレータとして利用することができる．

2.2.6. テクスチャ

描画する図形の色（光の反射率）などの表面材質を画像で制御するテクスチャマッピングの機能は，1 次元・2 次元・3 次元の定数配列として利用できる．

これらはテクスチャなので，一般的な配列とは異なり，アドレッシングを実数値で行うことができる．フィルタに線形補間を指定しておけば，テクスチャの画素の中間的な位置からは近傍の画素値からの補間値を得ることができる．また，テクスチャの周辺部分から値を取得する場合も自動的にクランプされるため，テクスチャからはみ出た部分に対して特別な処置を行う必要がない．

バーテックスシェーダでは並行動作する他のシェーダプロセッサに入力された頂点属性を知る手立てはないが，頂点バッファオブジェクトをテクスチャバッファオブジェクト (Texture Buffer Object, TBO) として参照すれば，頂点のインデックス（整数値）を用いて他の頂点に入力された頂点属性を参照することができる．

2.2.7. コンピュートシェーダ

バーテックスシェーダやフラグメントシェーダを使って数値計算を行う場合は，描画命令の発行によって処理を開始する必要がある．数値計算を図形描画と同期して行う場合はそれで問題ないが，非同期に処理を行いたい場合もある．また，ラスタライザの起動やフレームバッファへの書き込みのコストは少なくないため，それらの機能を活用するのでなければ，それらはできるだけ避けたいという要求もある．

コンピュートシェーダはデータの入力と出力にテクスチャを用いるが，それらを実数値でアクセスするテクスチャとしてではなく，要素を整数値で指定するイメージとして利用する．計算結果もこのイメージに格納する．また実行は描画命令によらず，コンピュートシェーダ単独でディスパッチすることになる．

3. 応用例

3.1. 点群処理

3.1.1. トランスフォームフィードバック

ばね質点モデルを陽解法により解くような場合は，現在の質点位置から，その次のタイムステップにおける位置を求め，現在の質点位置を更新する必要がある．

この場合は更新した質点位置をトランスフォームフィードバックにより頂点バッファに書き戻し，その頂点属性を次のタイムステップにおけるバーテックスシェーダの入力として用いる[2].

また，ばね質点モデルでは，ばねで接続された先の質点の運動も知る必要がある．その頂点属性は，テクスチャバッファオブジェクトを用いて参照することができる[3].

3.1.2. テッセレーション

Microsoft Kinect などの RGB-D カメラで得られる深度情報は，深度が計測できなかったところのデータが欠損している．このため，この深度データ（点群データ）を加工せずに立体形状を生成しようとすると，データの欠損部分に不正な形状が現れる．

そこでテッセレーションにより不正な形状のポリゴンを破棄したり，欠損部分を埋めるポリゴンを生成したりすることができる．

3.2. フィルタ

3.2.1. フィルタ処理

以下のプログラムは，OpenGL のシェーディング言語 GLSL のフラグメントシェーダにおいて，テクスチャからのデータ取得（サンプリング）を行う例である．これは image に割り当てられたテクスチャの，処理対象の画素の位置 gl_FragCoord.xy と同じ位置のデータ（色）を取得して，フレームバッファに出力するものである．これはテクスチャの画像をそのまま表示領域に表示する．

```
#version 330
uniform sampler2DRect image;
layout (location = 0) out vec4 fc;
void main(void)
{
  fc = texture(image, gl_FragCoord.xy);
}
```

ここで fc の行を次のように書き換えると，テクスチャを x 方向に微分した画像が表示される．テクスチャから取得したデータから，そのデータよりも x 方向に 1 画素分 ivec2(1, 0) ずれたところから取得したデータを引くものである．

```
fc = abs(texture(image, gl_FragCoord.xy)
 - textureOffset(image, gl_FragCoord.xy,
   ivec2(1, 0)));
```

3.2.2. 中間値フィルタ

中間値フィルタではデータの並べ替えが必要になるが，分岐や判断が繰り返される一般的な並べ替えのアルゴリズムは，シェーダプロセッサは効率的に実行することができない．そこで，三点の並べ替えをインラインで繰り返す手法などが用いられている．

4. おわりに

ここで示した活用方法は，グラフィックスでの応用に近い，非常に基本的なものである．グラフィックスへの応用であっても，これ以外にも単にポリゴンを投影してレンダリングするのでないボリュームレンダリング手法やメタボールのレンダリングのほか，画像処理におけるステレオマッチングや形状再構成，あるいは機械学習や強化学習での利用など，応用は数限りない．

GPU のハードウェアの機能をグラフィックス処理にとらわれずに解釈することによって，GPU の活用方法をさらに広げられると考える．

参考文献

[1] The Khronos Group Inc. “OpenGL Overview,” https://www.opengl.org/about/ (2018年2月25日参照)

[2] 床井研究室. “ゴムシミュレータ（１）,” http://marina.sys.wakayama-u.ac.jp/~tokoi/?date=20111129 (2018年2月25日参照)

[3] 床井研究室. “ゴムシミュレータ（２）,” http://marina.sys.wakayama-u.ac.jp/~tokoi/?date=20111207 (2018年2月25日参照)

[4] 床井研究室. “GLSL で画像処理 (2) 簡単なフィルタ,” http://marina.sys.wakayama-u.ac.jp/~tokoi/?date=20140726 (2018年2月25日参照)

[5] GitHub, “image0/median3.frag at master · tokoik/image0,” https://github.com/tokoik/image0/blob/master/median3.frag