3K-6

GPGPUによるグラフィックスツール間の変換処理

渡邉 優太† 坂下 善彦‡
湘南工科大学大学院 工学研究科 電気情報工学専攻†
湘南工科大学 工学部 情報工学科‡

1 はじめに

3DCG(Three-dimensional Computer Graphics)ツール間のデータ形式は各ツールの開発経緯の違いから統合されることなく、各ツールがプラグインを備える形でデータ変換を行っている。データには頂点座標やマテリアルなどで構成されており、コンテンツの要素数が増加すると共に比例して情報量は増大する。その為、構文解析器(Parser)を用いて 3DCG ツール間のデータ変換に掛かる時間も増加する。我々は可視化システムの構築を行っており、可視化対象となるデータを即時に映像として反映させるシステムを目指している。その為、モデルをレンダラーへ渡す I/O の高速化が重要になる。そこで本稿では 3DCG モデルにおける Parser を並列処理する手法を考案し、3DCG ツール間の変換処理の高速化を図った。本稿では GPU(Graphics Processing Unit)を汎用的な計算に用いる技術である、GPGPU(General-Purpose Computing on GPUs)を用いて 3DCG における Parser の並列処理手法を考案した。ここではプラットフォームに依存しない並列処理環境を提供するために、OpenCL(Open Computing Language)での実装を検討した。

2 変換処理

3DCG データは形式言語[1]で定義される。同一の形状を表すデータをそれぞれ別の 3DCG の形式言語で書き表した例を図 1 に示す。

```
Object "Obj name" {
    vertex n {
        X Y Z ・・・ V0
                :
        X Y Z ・・・ V3
    }
    face m {
        4V(V0 V1 V2 V3) M(mat_num) UV(・・・)
    }
}
```

```
mesh {
  smooth_triangle {
    <V0>, <Norm_Vec0>, <V1>, <Norm_Vec1>, <V2>, <Norm_Vec2>
    <V0>, <Norm_Vec0>, <V2>, <Norm_Vec2>, <V3>, <Norm_Vec3>
    texture { mat_name }
  }
}
```

図 1. データ定義の一例

図 1 の場合にはデータを定義する形式の名称や性質の違いがある。上部の形式では 4 角形の面で形状を表現している。しかし、下部の形式では 3 角形の面にし、形状の表現を行っている。その為、データの書き換えが必要となる。形式言語の場合は有限オートマトン(FA)[2]によって構文解析を行う。その例を図 2 に示す。図 2 にあるΣは文字列の有限集合となり、Num は読み取る値が数字である場合の遷移である。ここでは Num に記述される数字が変換に必要なパラメータとなり、この例では U, V, (, Num,)という順番で入力された場合にこの FA では受理される。そして受理されたデータから別の形式言語に変換する。また、存在しない要素などある、場合デフォルト値を定めてデータに付加している。最後に変換処理の全体の手順を以下に示す。

1) Parser によって構文解析を行う
2) FA によって受理されたものからパラメータ情報を得て、必要であればデータを加工する
3) パラメータ情報を変換先形式へ当てはめる

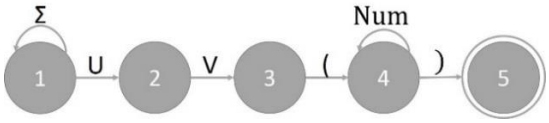

図 2. 有限オートマトンの例

3 提案手法

3DCG データの構造の図 3 に示す。データ構造は木構造になっており、葉の部分をチャンクと呼ぶ。図 3 ではマテリアルやオブジェクトなどがチャンクに相当する。

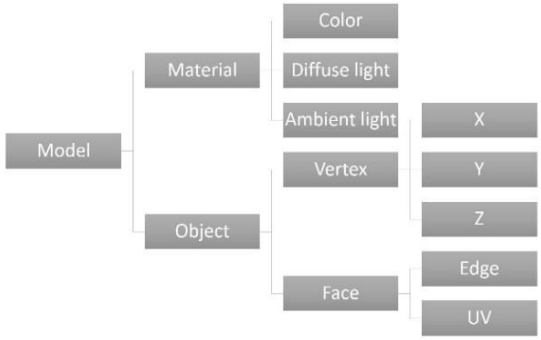

図 3. 3DCG データ構造

3DCG データは文字列の集合であり、その中から図 3 にある vertex の頂点座標や face のメッシュ情報、マテリアル情報、UV 値などを Parser が理解する。形式言語で記述されたパラメータ情報を Parser が構文解析を行い理解する。そして変換先が理解できる形に変換を行う。本稿は構文解析に用いられる Parser を GPU 上に実装し、並列化によって高速化を図る。その手法としてボイヤームーア(BM)法[3]を用いてチャンクの検索し、先頭からの位置(オフセット)を特定する。オフセットを特定する理由は

The data transformation method processed by GPGPU for data transfer between the graphics tools
† Shonan Institute of Technology, Graduate School
‡ Shonan Institute of Technology, Department of Information Sience

チャンクの種類毎に FA は異なり、チャンクの位置を検知する為に全体の文字列から各チャンクのオフセット量を得る。そして、オフセットから次のオフセット間でチャンクに対応する FA による構文解析を行う。

4 並列処理

4.1 OpenCL

OpenCL[4]とは様々なアーキテクチャ(CPU, GPU, DSP 等)で並列処理(SIMT : Single Instruction Multiple Thread)を行う為のフレームワークである。NVIDIA 製の GPU を用いる場合、CUDA を用いるのが一般的である。しかし、全ての環境が NVIDIA 製の GPU を用いているわけではない為、様々な環境で動作する必要がある。そこでクロスプラットフォームで動作し、ソースレベルで互換性のある OpenCL を用いて行った。OpenCL のメモリモデルは図 4 のようになる。名称は違うが CUDA と似たメモリモデルであり、図中のメモリは上部にあるほど高速にアクセスすることが可能となる。また、CUDA と同様にホストコードとデバイスコードに別れる。Work Group 間ではそれぞれ別のサブルーチンが動いており、その Work Group 内ではスレッドを複数生成して Work Item でサブルーチンが実行される。これらは非同期に実行される。

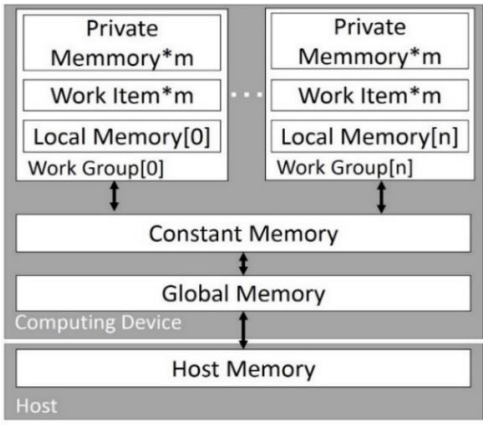

図 4. OpenCL メモリモデル

4.2 並列化手法

BM 法による並列化を図 5 に示す。本稿では文字列を Work Group にブロックに分けた状態で分割する。このブロック分割はシンタックスが途切れてはいけない。その為、改行等でブロック分割を行う。

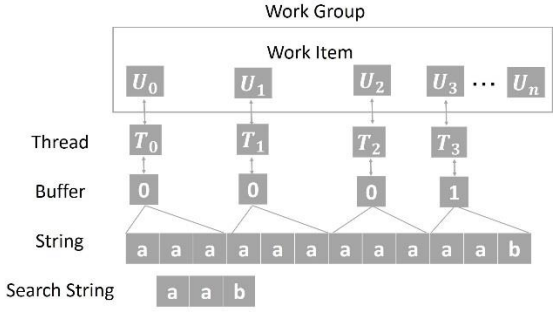

図 5. BM 法による並列化

そしてブロック文字列を BM 法による文字列検索の並列手法を図 5 に示す。ブロック内に存在する文字列をシフト量に分割し、並列に行っている。本稿で扱うデータの Parser 並列手法を図 6 示す。本稿では分割した文字列各計算ユニットに与え、FA によって構文解析を行う。

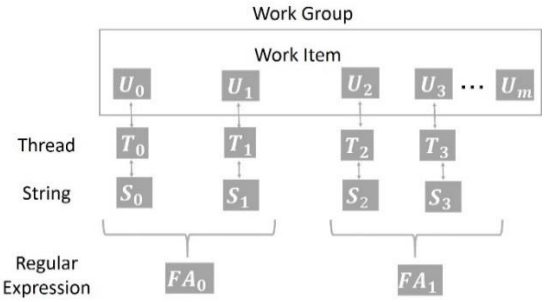

図 6.有限オートマトンの並列化

4.3 課題

3 章で述べた手法を用いて、記述されているチャンクのオフセットを特定し、FA によってパラメータを拾い出す。しかし、速度にかかわる問題が存在する。文字列の情報から事前にマッチ数を正確に見積もることが不可能である。更に、OpenCL ではデバイス上のメモリを動的確保することはできない。その為、メモリを余分に確保することになるが、その場合にデバイス上のメモリからホストのメモリへの転送オーバーヘッドは大きくなる。これは OpenCL、CUDA 問わず検索速度低下の大きな要因になる。これに対し本研究では文字列データの改行数を調べ、その数だけメモリの確保を行っている。これは改行によって要素の区別を我々のデータは行っている。これにより無駄は存在するが、バッファオーバーフローの心配はなくなる。

5 まとめ

3DCG において、ラスタライズする上で重要な文字列パラメータを OpenCL 環境で取得する手法を提案した。しかし、文字列内に存在する要素数や検索するパターンの数量によって、高速になる場合と低速になる場合があると考えられる。更に検討を進め、各種のデータによる変換処理を行い、高速化を追求する。

参考文献

[1] Chomsky, Noam, "Three Models for the Description of Language," IRE Transactions on Information Theory, Vol. 2 No. 2, pp. 113-123, 1956.
[2] 石田清, “アルゴリズムとデータ構造”, 文字列のアルゴリズム, 第 5 章, 岩場書店, 1989.
[3] R. S. Boyer and J. S. Moore, “A Fast String Searching Algorithm”, Communication of the ACM, Vol.20, pp.762-777 1977.
[4] OpenCL specification 1.1, <http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf>.