■2 群（画像・音・言語）－3 編（コンピュータグラフィックス）

4 章 コンピュータグラフィックスハードウェア

（執筆者：張　英夏・中嶋正之）［2012 年 10 月 受領］

■概要■

近年，よりリアリスティックなレンダリングを可能にするために，入力データ及び個々のレンダリング処理が非常に複雑になってきた．こうした状況に応じて，ハードウェアも飛躍的に進歩してきた．本章では，こうしたコンピュータグラフィックス向けの 3 次元入出力装置について説明する．

【本章の構成】

4-1 節では，出力（描画）用ハードウェア，中でも近年非常に注目を浴びているグラフィックスプロセッシングユニットについて解説する．4-2 節では，3 次元入力のためのハードウェアについて解説する．

■2 群 － 3 編 － 4 章

4-1 グラフィックスハードウェア

（執筆者：張　英夏・中嶋正之）［2012 年 10 月 受領］

3 次元コンピュータグラフィックスをレンダリングするためには非常に多くの演算を必要とする．従来は CPU（Central Processing Unit）上でそれらの計算を行っており，計算コストが非常に高いという問題があった．そのために，産業分野などで 3 次元レンダリングが必要な場合には SGI（Silicon Graphics Interface）など，3 次元演算専用のハードウェアを搭載した高価なワークステーションを用いていた．

一方，1999 年 8 月に NVIDIA から GeForce 256 という，パーソナルコンピュータ（PC）向けグラフィックスボードが発表された．これは，PC 上で 1500 万ポリゴン/秒及び 4 億 8000 万ピクセル/秒の描画速度を実現した．NVIDIA は，このようなグラフィック処理を行うハードウェアを GPU（Graphic Processing Unit）と呼ぶようになった．NVIDIA のサイトによると，GPU は技術的に「1000 万ポリゴン/秒以上を処理できる，変換，ライティング，トライアングルセットアップ及びクリッピング，レンダリングなどの各エンジンを 1 枚のチップに統合したシングルチッププロセッサ」と定義づけている [1]．GPU はその後，プログラマブルシェーダに対応していき，更には汎用的なプログラムを動かせられるようになったことで（GPGPU：General Purpose GPU），グラフィックス分野のみでなく，様々な分野から注目されることとなった．メニーコアをもつ GPU は，超並列プログラミングを可能にしてくれるため，その処理速度の速さが最大の魅力である．こうした低価格での高速処理に着目し，更に 2009 年には長崎大学で 256 台の GPU をつなぎ合わせた超低額スーパーコンピュータ DEGIMA を開発し，同年の ACM ゴードンベル賞（ACM Gordon Bell Prize）を獲得している [2]．

GPU プログラミングを可能にする開発環境として，物理シミュレーションを得意とする PhysX [3]，NVIDIA が提供する GPU 向け統合開発環境である CUDA（Compute Unified Device Architecture）[4]，様々なプラットフォームの言語を共通化しようと試みている OpenCL（Open Computing Language）[5] などが有名である．

本節では，NVIDIA 社の GPU のアーキテクチャ及び CUDA を中心に説明する．なお，GPU は正確には画像処理用の LSI（Large Scale Integration）チップを意味するが，その LSI チップが搭載されているグラフィックスボードを GPU と呼ぶことも多い．本節では，チップそのものは GPU チップ，グラフィックスボード全体は GPU と記す．

4-1-1 GPU のアーキテクチャ

現在，パーソナルコンピュータ向けグラフィックスボード製品として，AMD 社の Radeon シリーズと，NVIDIA 社の GeForce 系が主流となっている．本項では NVIDIA 社の GeForce 系グラフィックスボードの中で，2012 年 9 月時点で最新のアーキテクチャ（Architecture）である Kepler GK110 について説明する．

グラフィックスボードには，メモリと GPU チップ，そのほかビデオ入出力などのためのインタフェース，冷却ファン，補助電源（電源インタフェース）などが搭載されている．この中で，核となる GPU チップの構造について説明する．**図 4･1** に Kepler GK110 の模式図を示す．

図 4・1　Kepler GK110 チップ全体の模式図（NVIDIA 社提供）

まず，チップ全体に緑の四角が数多く並んでいるのが見える．これがアーキテクチャの最小単位である，CUDA コア（CUDA Core：以前は SP（Streaming Processor）と呼ばれていたもの）と呼ばれる演算装置である．CPU コアは 1 チップに高々16 個しかないのに比べ，Kepler GK 10 チップ上には最大 192×15 個の CUDA コアが搭載されている．

このCPUコア 192 個と，そのほかに倍精度ユニット 64 個，特殊関数ユニット 32 個，ロード/ストアユニット 32 個が集まって，もう一つ上の単位であるSMX（Streaming Multi-processor eXtreme）を構成している．SMXの主な役割は，適切な時間にCUDAコアに命令を配分し，実行させることである [*1]．そしてこのSMX 5 個とメモリコントローラ 6 個，L2 Cacheによって GPUチップが構成されている．なお，一つのCUDAコアは複数のスレッドを計算することができるため，全体としては非常に多くのスレッドを実行できる．そのため，スレッドは階層的に管理される．スレッドをまとめたものがブロックという概念，ブロックをまとめたものがグリッドという概念になる．

[*1] 前世代である NVIDIA Fermi までは，SM（Fermi までは SMX ではなく，SM と呼んでいた）の中ですべて命令スケジューリングを行っており，複数の命令を同時実行させるために，スレッド数によっては膨大なリソースを必要としていた．Kepler では SMX における処理を簡素化した．具体的には，CPU のドライバソフトウェアの中に演算命令のレイテンシをチェックしたり命令間の依存性をチェックする仕組みを設け，GPU 命令の中にその情報を埋め込み，SMX に渡す．SMX では埋め込まれた情報から実行可能になった命令を選び，命令をデコードし，CUDA コアに渡すというものである．このように命令スケジューリング部分を簡素化することで，SMX をスリム化でき，そのスリム化された部分に CUDA コアを数多く詰め込むことで巨大なメニーコアシステムが可能となった．

4-1-2 GPU とメモリ

GPU には，グラフィックスボード上のメモリ（オフチップメモリ）とチップ上のメモリ（オンチップメモリ）がある．この中で，CPU 側からはオフチップメモリにアクセスすることができ，領域の確保や開放，データのコピーなどを行うことができる．

Kepler におけるメモリ階層は**図 4･2** の通りである．

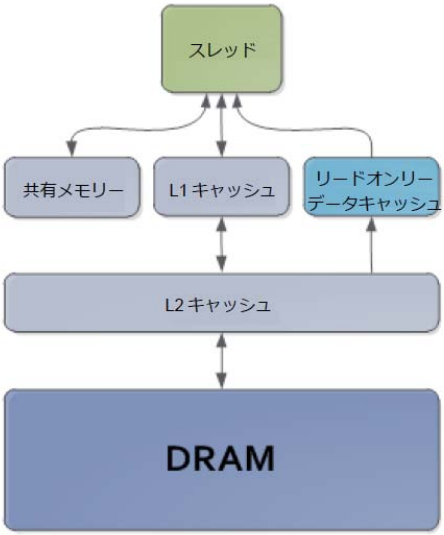

図 4・2　Kepler におけるメモリ階層（NVIDIA 社提供）

各々のメモリは以下のような特徴をもっている．

- DRAM：オフチップメモリ．大容量だが GPU 側からのアクセス速度が遅い．CPU コードと GPU コード両方が書き換えることができる．
- L2 キャッシュ：オンチップメモリ．図 4･1 の真ん中に存在している共有メモリである．比較的高速にアクセスすることができるが，容量はそれほど大きくない．
- 共有メモリ，L1 キャッシュ：オンチップメモリであり，特にコアの最も近くに存在するメモリで高速にアクセスすることができる．しかしながら，容量は小さい．
- リードオンリーデータキャッシュ：オンチップメモリであり，共有メモリや L1 キャッシュと同じで SMX 内に存在する．予め定数として分かっているものはこちらのキャッシュを使うことによって，共有メモリや L1 キャッシュの負荷を軽減することができる．

実際のプログラムでは，必要とする容量やアクセス速度によってメモリを使い分ける必要がある．

4-1-3 GPU プログラムの処理の流れ

本項では，GPU を利用したプログラミングの中で，特に CUDA に限定して述べる．なお，CUDA のツールキット及びドライバ，SDK（Software Development Kit）は NVIDIA のサイトから無料でダウンロードすることができる [4]．現在，CUDA に関して様々な参考書が出版されており，詳しい実例とともに解説されているので，そちらも参考にされたい [6)~8)]．

CUDA とは GPU コンピューティング向けの統合開発環境で，コンパイラ，ライブラリ，デバッガなどから構成されている．C 言語をベースにしているため，C 言語を修得している人であれば分かり易い．なお現在は，Java や Phyton などから CUDA を利用するためのバインティングも存在する [9), 10)]．

CUDA の特徴としては，C 言語がベースなのでプログラミングを勉強したことがある人であれば馴染み深い点，Windows や Linux, Mac OS X など様々なプラットフォームで動作する点，フリーで利用できる点などをあげられる．

CUDAプログラムの実行時の流れは以下のようである．まず，予め作成されたCUDAプログラム（拡張子は.cu）をnvccコマンドによってコンパイルする．コンパイル時には，GPU呼び出し記述や各種設定を参考に，CPU向けコンパイル，GPU向けコンパイルを別々に行った後，それぞれを結合し最終的な実行ファイルを作成する．実行ファイルを実行すると，まずはCPU側（ホスト側と呼ばれる）でプログラムが開始され，GPU側（デバイス側と呼ばれる）で動作すべきプログラム（カーネル）をGPUにロードする．次に，GPU側で処理すべき部分に必要となるデータを用意し，それをデバイス側にコピーする．その後，デバイス側でデータ処理が行われ，その処理結果をホスト側にコピーする．最後にその結果についてホスト側で処理を行い，プログラムが終了する [*2]．

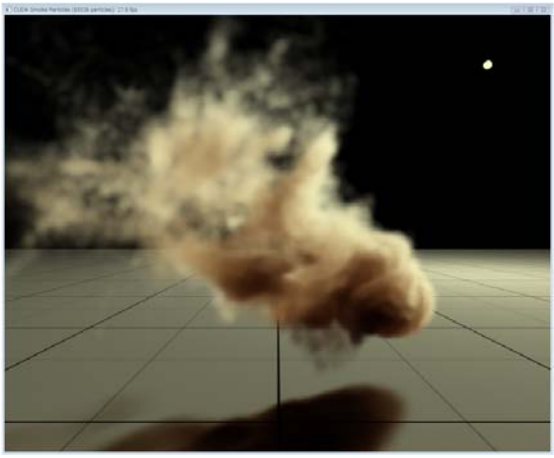

図 4・3　CUDA SDK に付属しているサンプルプログラムの実行例

[*2] なお，Fermi までの世代では CPU 側のみがカーネルを呼び出せたため，再帰などを行う場合でも一処理ごとの間に通信を行う必要があり，非効率的であった．Kepler GK110 では GPU 内で演算結果の依存性を認識し，ループや再帰を自動処理することができる，Dynamic Parallelism という仕組みが可能になっている．

なお，GPU を導入し CUDA プログラミングを行う際の注意点は以下の通りである．CPU と GPU は別々のメモリを利用しており，CPU と GPU 間でデータをコピーする必要があるが，意外と時間がかかる．このオーバーロードを考慮しても CUDA を利用することに利点があるか検討する必要がある．また，メモリの階層によってアクセス速度が異なるので，何をどのメモリに保存するかを熟慮し，グローバルメモリ（DRAM）アクセスを減らす必要がある．

■2 群 － 3 編 － 4 章

4-2 3 次元入力

（執筆者：張　英夏・中嶋正之）［2012 年 10 月 受領］

3 次元入力は，静的なターゲットについて形状の再構築を行う場合と，動作をターゲットとすることで 3 次元動作入力を行う場合がある．また，一般的なカメラやビデオカメラを用いる方法，距離センサやモーションキャプチャシステムなどの特殊なデバイスを用いた方法がある．本節では，使用デバイスによって分類を行い，手法を説明する．

4-2-1 一般的なカメラやビデオを用いる方法

最も単純な方法に，視体積交差法と呼ばれる方法がある [11]．

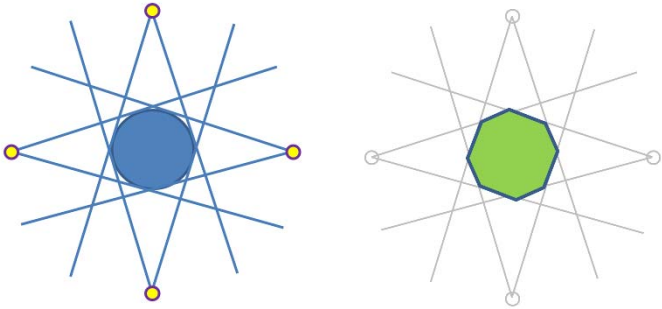

図 4・4　視体積交差法

まずは**図 4･4** 左に示すように，複数台のカメラを対象物の周りを囲むように配置し，同時に映像を記録する．次に，各カメラ位置から見た対象物体の輪郭を元に，対象物体が存在し得る部分空間を求める．これを視体積と呼ぶ．この視体積を各視点毎に求め，図 4･4 右に示すようにそれらの交差する部分を抜き出すことにより，3 次元形状を求めることができる．この方法では，カメラの台数が充分にあればある程度の精度が期待できる．また，色恒常性を利用して更に精密な 3 次元形状を求めることも可能である [12]．

そのほか，ストライプやグリッドなどのパターン（コード化パターン光：Coded Structured Light）を投影し，パターンの歪みから 3 次元形状を推測し復元する手法 [13]，複数の写真を用いて 3 次元を再構築（視点移動画像を生成）する方法 [14]，ビデオカメラを用いてフレーム毎の特徴点マッチングを行うことで 3 次元を復元する手法 [15] などがある．

4-2-2 ハプティックスデバイスを用いた 3 次元入力

ハプティクスデバイス（Haptics Device：力学提示装置）とは，ユーザからの 3 次元入力に対し，力のフィードバックを与える装置を意味する．これは，単なる 3 次元入力ではなく，それに対するフィードバックを与えられるため，トレーニングシステムや遠隔操作システム，ゲームなど様々なインタラクションを要する分野で用いられている．広く利用されている商用ハプティックスデバイスとして PHANTOM [15] が有名である．その他，SPIDAR [17] などがある．

4-2-3 モーションキャプチャシステム

モーションキャプチャシステム（Motion Capture System）とは，現実の人間や物体に複数のセンサをつけ，その動きをデジタル的に記録するための装置である（**図 4･5**）.

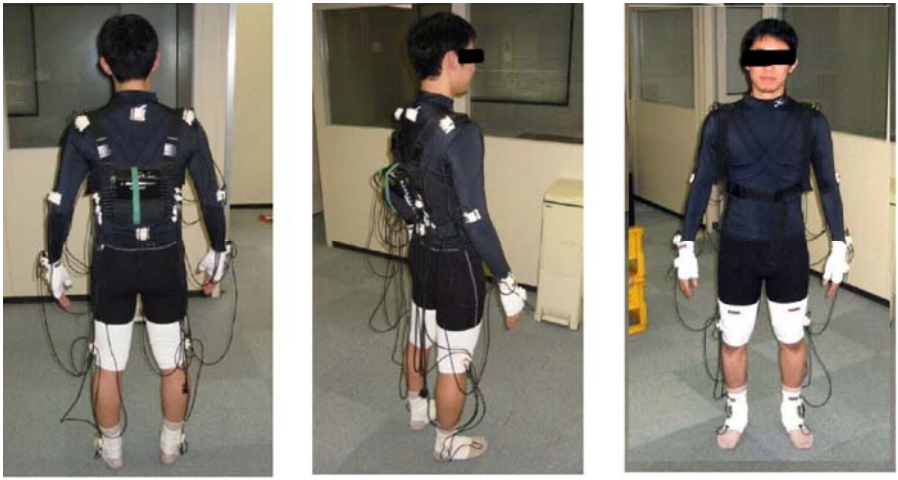

図 4・5　モーションキャプチャシステムの装着例

測定されたセンサの位置情報に基づき，姿勢を推定することができる．この推定された姿勢をボーンモデルに適用し，そこにキャラクタのポリゴンをフィッティングすることによってキャラクタアニメーションを生成することができる．モーションキャプチャシステムは大きく分けて光学式，磁気式，機械式の 3 種類がある.

光学式のモーションキャプチャシステム（Optical Motion Capture System）では，反射式のマーカまたは赤外線発光ダイオードを用いる．このマーカの位置は周辺に設置された複数のカメラによって追跡される．この方式は最も自由度の高い動作測定が行えるという利点があるが，マーカの位置がカメラの死角に入ってしまうと，動きを追跡できないという問題点がある.

磁気式のモーションキャプチャシステム（Magnetic Motion Capture System）ではコイルを内蔵したセンサを用いる．動作測定を行うために磁界を発生させ，移動した時にセンサから生じる誘導起電力を用いて 3 次元位置を測定する方式である．しかしながら，測定環境の磁場の影響を大きく受けるため，金属などを測定範囲から遠ざける，小さな動きに限定し均一な磁場を確保するなどの工夫が必要となる.

機械式のモーションキャプチャシステムでは加速度センサやジャイロセンサを直接関節に取り付ける．体全体に機械的な計測器を取り付ける必要があり，動きが非常に制限される.

計測された動作データは 3 次元モデルに適用される．通常は動作取得時の対象物と同じ構造のモデルに適用するが，体のパーツの長さ比率が異なるキャラクタにも違和感なく適用する手法も提案されている [18].

4-2-4 新たな 3 次元入力システム

4-2-3 項で述べた通り，従来の機械式は撮影環境には影響を受けないが，大がかりな計測器を取り付けるためにアクタの動きが制限されるという問題があった．これを解決する方法として，複数の慣性センサが内蔵されたスーツを着用することでモーションキャプチャを行う方法が提案されている [16]．これは，スーツの中 17 箇所に小型慣性センサを搭載し，そのセンサがアクタの動きをリアルタイムに検出, PC に送信することによってモーションキャプチャが得られる．カメラやスタジオを必要とせず，また装置が軽量であるためアクタに対する動作制限もないことから狭い屋内や屋外でも使用できる．

以上でモーションキャプチャシステムについて概観した．いままで述べたシステムは高価なものが多く，一般ユーザが手軽に使えるものではない．そこで，最後にいくつか安価なシステムを紹介する．

まずは Microsoft のゲーム機 Xbox のコントローラーである Kinect をあげる．Kinect には RGB カメラと深度センサが内蔵されており，深度センサ（赤外線カメラ）を用いて被写体からカメラまでの距離を計測することが可能である．このため，マーカを用いずにリアルタイムモーションキャプチャが可能である（**図 4･6**）．更に，2011 年には KinectFusion [17] と呼ばれる技術が開発され，モーションキャプチャのみならず，シーンの 3 次元再構築も可能になった．これは，Kinect から取得された 3 次元点群画像群を，ICP アルゴリズムを利用してレジストレーションすることによって，3 次元シーンをリアルタイムで再構築している．現在，このクローンプロジェクトとして PCL（Point Cloud Library）[18] で Kinfu という名前のモジュールが実装されており，ソースコードがダウンロード可能である（**図 4･7**）．更に，メモリ使用に工夫を行い大規模シーンの再構築を可能にした Kintinuous [19] なども開発されるなど，Kinect を用いた 3 次元再構築技術は，現在活発に開発されている．

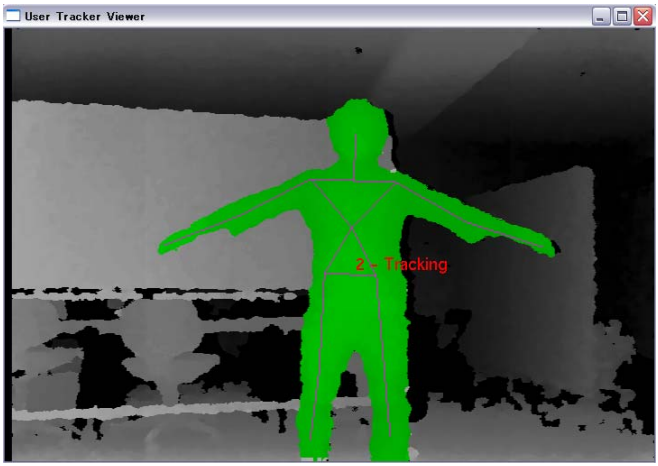

図 4･6　kinect によるリアルタイムボーンモデル抽出結果
（東京都市大学画像工学研究室冨樫晃己氏提供）

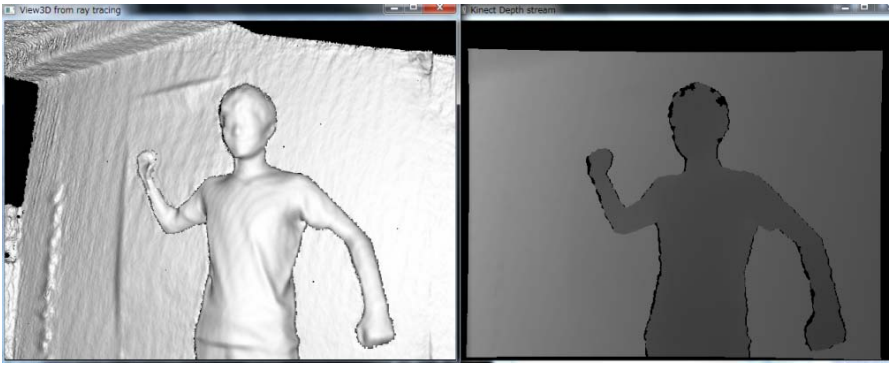

図 4・7 kinect によってリアルタイムで 3 次元を再構築した結果．左が再構築された 3 次元データをシェーディングしたもの，右が取得したデプスデータ．
（東京都市大学画像工学研究室冨樫晃己氏提供）

次に，モーションをキャプチャするのではなく，小型の人形を動かして直感的にポーズを入力することができるシステム [20] がある．これは Qumarion という人型入力デバイスを利用する．この入力デバイスは従来の関節フィギュアと同等の形をしており，関節部分を折り曲げてポーズを付ける．稼働関節数は 16 個であり，合計で 32 のセンサが装着されている．各関節からのデータは毎秒約 120 フレームで PC に取り込まれ，リアルタイムに 3 次元キャラクタに反映することができる．

■参考文献

1) NVIDIA. [Online]. http://www.nvidia.co.jp/page/geforce256.html
2) 長崎大学工学部先端計算研究センター. [Online]. http://nacc.nagasaki-u.ac.jp
3) NVIDIA PhysX. [Online]. http://www.nvidia.co.jp/object/physx_new_jp.html
4) NVIDIA CUDA. [Online]. http://developer.nvidia.com/cuda/cuda-downloads
5) OpenCL. [Online]. http://www.khronos.org/opencl/
6) 額田　彰, 青木尊之, “はじめての CUDA プログラミング,” 工学社, 2009.
7) Wen-mei W. Hwu David B. Kirk, “CUDA プログラミング実践講座,” ボーンデジタル, 2010.
8) 岡田賢治, 小山田耕二, “CUDA 高速 GPU プログラミング入門,” 秀和システム, 2010.
9) PyCUDA. [Online]. http://mathema.tician.de/software/pycuda
10) JCuda. [Online]. http://www.jcuda.org/jcuda/JCuda.html
11) W. N. Martin and J. K. Aggarwal, “Volumetric description of objects from multiple view,” IEEE Trans. PAMI, vol.5, no.2, pp.150-159, 1987.
12) S. Seitz and C. Dyer, “Photorealistic scene reconstruction by voxel coloring,” IJCV, vol.35, no.2, pp.151-173, 1999.
13) J. Salvi, J. Pages, and J. Batlle, “Pattern codification strategies in structured light systems,” Pattern Recognition, vol.37, no.4, pp.827-849, 2004.
14) M. Geosele et al., “Ambient point clouds for view interpolation,” ACM Transactions on Graphics, vol.29, no.4, pp.95:1-95:6, 2010.
15) PHANTOM. [Online]. http://www.ddd.co.jp/phantom/
16) MVN. [Online]. http://www.0c7.co.jp/products/mvn/
17) Richard A. Newcombe et al., “KinectFusion: Real-time dense surface mapping and tracking,” Proceedings of

the 2011 10th IEEE International Symposium on Mixed and Augmented Reality, pp.127-136, 2011.
18) Point Cloud Library. Point Cloud Library. [Online]. http://pointclouds.org
19) Thomas Whelan et al., “Kintinuous: Spatially extended KinectFusion,” RGB-D Workshop at RSS, 2012.
20) SoftEther. [Online]. http://quma.jp/jp/quma/