はじめに

本書では，Graphics Processing Unit（GPU）と呼ばれるLSIを用いて，図形や立体に関する複雑な計算を高速処理するためのプログラミング手法を解説する。GPUはその名前が示すように，本来はコンピュータにおけるグラフィックス処理を担当する専用LSIであり，市販の多くのパーソナルコンピュータ（PC）にすでに搭載されている。このGPUは性能向上が著しく，「2年間で2倍の性能向上」というムーアの法則を上回るペースで進歩を続けている。**図1**に，GPUメーカとして知られるnVIDIA社が，自社のGPUと多くのPCに採用されているインテル社のCPUの単精度浮動小数点演算の性能変化をグラフ化したものを示す［1］。GPUの性能向上のペースがCPUを凌駕していること，そして現時点ですでに10倍近い性能差があることがわかる。

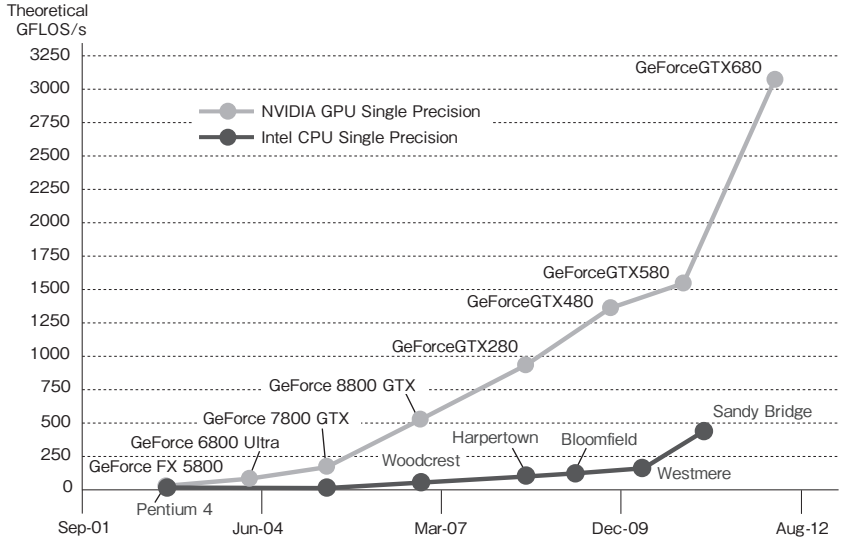

図1　CPUとGPUでの単精度浮動小数点演算の処理速度の比較（[1] より）

このGPUの性能に目を付け，これをグラフィックス処理だけでなく，もっと汎用的な計算で利用しようという動きが世界中で活発化している。このような流れをGPGPU（General-Purpose computing on GPU）と呼ぶ。近年ではスーパーコンピュータのハードウェアにGPUを取り込むことも盛んに行われている。たとえば2010年にスーパーコンピュータの性能世界1位となった中国の「天河」には，7,168個のnVIDIA社製のGPUが搭載されている。現在のGPGPU技術は，大規模な行列計算や，分子動力学シミュレーションなどの数値解析の分野での利用が中心となっている。本書では，これまであまり取り上げられてこなかった，図形や立体に関する計算においてGPUを利用する手法を紹介する。本書を読めば，GPGPU技術が数値解析の高速化だけではないこと，そして少しの工夫で従来技術と比較して1桁以上高速な図形処理プログラムが実現できることがわかるだろう。

図形処理におけるGPUの利用法を説明する前に，GPUの高速性の理由について簡単に触れておこう。通常のCPUでは，処理性能を高めるために，複雑な制御機構やキャッシュと呼ばれる高速メモリが取り入れら

れており，多くのトランジスタがこれらの機能実現に割かれている。結果としてCPUには，比較的少数の計算ユニット（ALU，Arithmetic Logic Unit）しか用意することができない。一方GPUでは，意図的に制御機構をシンプルにし，またキャッシュメモリの量も減らし，その代わりLSIにできるだけ多くの計算ユニットを配置している。**図2**にはこの2つの違いを図で示した［1］。このようなアーキテクチャは，複雑な計算を1つだけ実行する場合には不利だが，同じ内容でしかも互いに独立な単純計算を多数実行する場合には，各ユニットに計算を割り当て全ユニットが並列に処理を行うことで，結果的に単位時間あたりの処理性能（これをスループットと呼ぶ）を高めることができる。このような考え方に基づいて設計された計算機アーキテクチャを，スループット指向（Throughput-Oriented）なアーキテクチャと呼ぶ［2］。GPUはこのスループット指向アーキテクチャの代表例と言える。

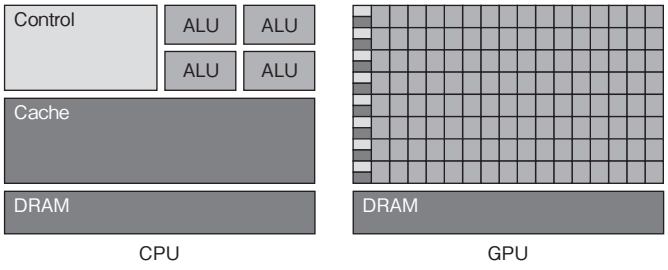

図2　CPUとGPUのアーキテクチャの比較（［1］より）

GPUによる図形処理

本書では，GPUの並列処理機能を利用して，特に2次元図形や3次元立体に関する複雑な図形計算を高速処理するプログラミング技術について述べる。グラフィックス処理用のハードウェアを用いて図形計算を高速化するというアイデアはかなり古くからあり,たとえば1986年にはCSGと呼ばれる立体モデリング手法のハードウェア処理の論文が発表されている［3］。この種の研究はその後もさまざまな研究者により継続的に進められ，立体の断面形状の生成［4］，ロボットの動作プランの生成［5］などの研究成果が知られている。その後，グラフィックス処理に特化したワークステーションの誕生やGPUの登場とともにこの種の研究も増加し，2000年代初頭には，本書でも紹介しているボロノイ図の生成［6］やオフセット処理［7］などの研究成果が次々に報告された。当時はまだGPUを用いたプログラムの開発環境が未整備だったため，多くの研究は，OpenGLなどのコンピュータグラフィックス用のライブラリの機能を流用する形で行われた。そのため手法の汎用性に問題があり,ライブラリの機能がうまく適合する問題は解けるが,そうでない問題には手が出ない,という状況であった。

GPUの並列処理性能に目を付け，これをもっと多くの問題解決で利用するための動きは，まずプログラマブル・シェーダ（Programmable Shader）の利用という形で始まった。グラフィックスライブラリには，立体の陰影表示（シェーディング）のためにいくつかの基本関数が用意されている。しかしこれらの用意されている機能だけでは，よりリアルな表示や，目的に応じた選択的な表示，強調表示などが困難なため，シェーディング処理をプログラミング可能にするための枠組みと，プログラミングのための簡易言語がグラフィックスライブラリに用意されるようになってきた。これをプログラマブル・シェーダ（言語）と呼ぶ。たとえばOpenGLの最近のバージョンにはGLSL，またDirectXと呼ばれるライブラリにはHLSLという言語が用意されている。これらの言語は，シェーディング用のプログラム作成を目的に設計されたものだが，その機能をうまく利用することで，より一般

的な図形処理プログラムを作成することもできる。たとえば文献［8］には，シェーダを利用して立体間の衝突検出プログラムを実装した例が紹介されている。しかしシェーディング用のプログラミング機能を用いて一般計算を行うには無理も多く，プログラマブル・シェーダを用いたGPGPUは普及したとは言い難い。

このような状況は，nVIDIA社が2008年に発表したCUDA（Compute Unified Device Architecture，クーダ）により一変することになった。CUDAはGPUの並列処理機能を，通常のC言語から利用するための統合的な処理環境であり，コンパイラやライブラリ，デバッガ，プロファイラなどから構成されている。CUDAはC言語の拡張となっており，すべての機能がnVIDIA社から無償で提供されている。したがってCプログラミングに関する基本的な知識と，nVIDIA社のGPUを搭載したPCがあれば（それがWindows，マッキントッシュ，Linuxを問わず），誰でもGPUの高性能を活かしたプログラム開発が可能となる。その後，他社のGPUでも利用可能な処理言語としてOpenCL（Open Computing Language）も発表され，GPGPUの開発環境がさらに整備されることになった。このような環境の変化とともにGPU利用はますます活発化しており，科学技術計算の分野では，GPUを用いた並列処理プログラミングに関する知識は，すでに必要不可欠なものになっている。

本書の内容

本書は4部構成となっており，その内容は以下のとおりである。

- **第1部　コンピュータグラフィックスの基礎**
  図形や立体形状の処理では，対象図形や処理結果を表示するために，コンピュータグラフィックスの知識が欠かせない。1章～3章では，業界標準のグラフィックスライブラリであるOpenGLと，ユーザインターフェイス構築のためのツールキットとして普及しているglutを利用したプログラミング手法について解説する。1章ではまずプログラミングの準備，特にglutのインストール方法や使い方について説明し，次の2章では2次元図形の表示，そして3章では3次元立体の表示について説明する。これらの章で説明した表示技術は，以後の章で紹介するプログラム作成の際に利用される。すでにOpenGLやglutについて十分な知識を有している読者は，これらの章は読み飛ばしても構わない。

- **第2部　OpenGLを用いた並列図形処理**
  前節でも述べたように，初期のGPUによる図形処理の研究では，OpenGLなどのグラフィックスライブラリの関数を流用して計算を行うことが一般的であった。特にデプスバッファと呼ばれる技術を利用した例が多数知られている。そこで4章と5章では，このOpenGLを使った並列図形処理の例として，ボロノイ図の計算とオフセット面の計算の2つを紹介する。CUDAなどのGPGPU用環境が普及した今となっては，OpenGLを使った図形計算は古くさい手法と思われるかもしれない。しかし上記2つの計算では，CUDAよりOpenGLを使ったほうがはるかに簡潔かつ高速なプログラムとなる。またCUDAを用いたプログラムはnVIDIA社製のGPUでしか稼働しないが，OpenGLのみを用いたプログラムであれば，ほとんどすべてのPCで動かすことができる。

- **第3部　CUDAの導入とOpenGLとの併用**
  6章からは，CUDAを用いた図形処理プログラミングの解説に入る。まず6章において簡単な行列計算を例にCUDAの基本的な利用法を学んだ上で，7章では粒子法計算，そして8章では拡散方程式の数値解法を例に，CUDAのプログラミング作法を学習していく。これら3つの例は，CUDAが最も力を発

揮する数値計算法として，これまでにも多くの参考書で取り上げられている。ただ本書では類書とは異なり，単なる数値計算にとどまらず，計算結果をOpenGLを利用して画面表示するところまで説明している点に特徴がある。特にOpenGLのバッファオブジェクトという手法を利用して，GPUによる計算結果を（CPUを経由せずに）そのままGPUで画面表示する手法について詳しく解説した。この技術を用いると，CUDAで数値計算をしながら，同時に計算の経過を画面でアニメーション表示することができる。

- **第4部　CUDAによる複雑な図形処理**
  最後の9章～12章では，複雑な3次元立体に関する図形計算をCUDAを用いてGPUで並列処理する手法について，メタボール法（9章），マーチングキューブ法（10章），立体間の衝突検出（12章）の3つを例に説明する。これらの図形計算はいずれも本質的に並列処理向きだが，要素数が多いと処理すべき図形の組み合わせが爆発的に増加し，単純なアルゴリズムではGPUを用いても高速な計算は望めない。このような場合には，明らかに計算が不要な図形の組み合わせを事前に除外し，意味のある組み合わせに限定して並列処理を適用することが効果的である。そのような目的では，空間的な格子構造を用いる手法と，階層的な包含立体を利用する手法の2つが常套手段となっている。本書ではメタボール法とマーチングキューブ法で格子構造の利用法を紹介し，衝突検出では階層的な包含立体の利用法を説明した。特に包含立体については独立した章（11章）を用意し，その処理手法をプログラムとともに示した。

本書で紹介したサンプルプログラムは，すべてホームページからダウンロード可能となっている。そこでプログラムの解説の際には，重要な部分のソースコードのみすべて掲載し，それ以外については掲載を一部省略した。是非プログラムをダウンロードし，その内容を確認しながら本書を読み進めていただきたい。プログラムのダウンロードの方法や利用法は巻末に記した。

本書を読む前に

本書では，事前に読者がC言語を用いたプログラミングについて基礎知識を有していることを仮定している。特に数値計算，メモリの確保，構造体の利用，ポインタの利用についての知識が欠かせない。またベクトルや行列計算，三角関数に関する基礎的な数学の知識も仮定している。なお本書ではかなり高度な図形処理を扱うが，いずれの章でも最初の数節で扱う技術の基本について説明を行うので，これらに関する知識は特に必要としない。またコンピュータグラフィックスについても，1章～3章で十分な説明を行うので，それらについての知識も仮定しない。

プログラム開発には，Windows上で稼働するVisual Studioを用いている。サンプルプログラムも，すべてVisual Studioのプロジェクトとしてまとめられているので注意してほしい。また本書の6章以降に掲載したプログラムは，最新のCUDA 5.5を用いて実装してある（CUDA 5.0でも動作確認している）。ただし用いている関数は基本的なものばかりなので，プログラムにわずかに手を加えるだけで古いCUDA 4.0 ～ 4.3に対応させることができるだろう。なおCUDA 5.0以降では，それまでのバージョンのCUDAとヘッダーファイルの定義が異なっているので，古いバージョンのCUDAで動かす場合には，プログラム中のヘッダーファイルの宣言に修正が必要となる。

参考文献

[1] CUDA C Programming Guide, nVIDIA, 2012.

[2] K.Fatahalian and M.Houston: A Closer Look at GPUs, Communications of the acm, Vol.51, No.10, 2008.

[3] J.R.Rossignac and A.A.G.Requicha: Depth-buffering display techniques for constructive solid geometry, IEEE Computer Graphics and Applications, Vol.6, No.9, 1986.

[4] J.Rossignac, A.Megahed, and B.Schneider: Interactive inspection of solids: cross-sections and interferences, Computer Graphics, Proc. SIGGRAPH 92, 1992.

[5] J.Lengyel, M.Reichert, B.R.Donald, and D.P.Greenberg: Real-time robot motion planning using rasterizing computer graphics hardware, Computer Graphics, Proc SIGGRAPH 90, 1990.

[6] K. E. Hoff III, T. Culver, J. Keyser, M. Lin, and D. Manocha: Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware, Computer Graphics, Proc. SIGGRAPH 99, 1999.

[7] M. Inui: Fast inverse offset computation using polygon rendering hardware, Computer-Aided Design, Vol. 35, No.2, 2003.

[8] 原田隆宏, 田中正幸, 越塚誠一, 河口洋一郎: GPUを用いたリアルタイム剛体シミュレーション, 情報処理学会グラフィックスとCAD研究会, 2007-CG-126, 2007.