芸術科学会論文誌 Vol.4 No.1 pp.13－19

GPU コンピューティングの動向と将来像

宮田 一乗[1]　高橋 誠史[2]　黒田 篤[3]
[1] 北陸先端科学技術大学院大学・知識科学教育研究センター
[2] 北陸先端科学技術大学院大学・知識科学研究科
〒923-1292 石川県能美郡辰口町旭台 1-1
[3] 株式会社ジースポート
〒111-0036 東京都台東区松が谷 1-9-12
E-mail: { miyata, masa-t }@jaist.ac.jp, kuroda@gsport.co.jp

概要 PC のグラフィックボード(GPU)の果たす役割は年々加速しており，CPU 中心の計算環境から GPU 中心の計算環境へと徐々に移行している．また，GPU の持つ機能をプログラムで利用できる環境が整ったことにより，GPU をグラフィック処理専用の CPU の補助プロセッサとしてではなく，並列計算機として応用する研究が活発化している．本論文では，GPU コンピューティングで何が可能となり，今後どのような発展が期待できるかを，具体的な例を挙げながら述べる．

キーワード GPU コンピューティング，グラフィックプロセッサ，シェーディング言語，プログラマブルシェーダ

State of the arts and future of GPU computing

Kazunori Miyata[1], Masafumi Takahashi[2], and Atsushi Kuroda[3]
[1]Center for Knowledge Science, Japan Advanced Institute of Science and Technology
[2] School of Knowledge Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-1292 Japan
[3] gsport, inc.
1-9-12 Matsugaya, Taito-ku, Tokyo, 111-0036 Japan

Abstract The roles which PC's graphic boards play are accelerating year after year, and it is shifting to the calculation environment based on GPUs gradually from the calculation environment based on CPUs. After the programming environments for GPU functions are established, the researches which are considering GPU as a parallel computer, not only as a coprocessor of CPU for graphic processing, are activating. This paper describes what become possible by GPU computing, and what will be expectable from now on, giving concrete examples.

Keywords GPU Computing, Graphic Processor, Shading Language, Programmable Shader

1. はじめに

PC のグラフィックボード(ビデオカードとも呼ばれる．以下，GPU と略す）の果たす役割が年々加速している．

一部のグラフィックチップメーカでは，半年ごとの新チップ発表を公言しており，CPU の進化スピードに匹敵する勢いで改良が重ねられてきている．また，数年前までは，3 次元 CG を用いたコンテンツ制作環境は高価なグラフィックワークステーションが主流であったが，PC の計算速度や GPU の性能向上により，最近では PC ベースでの制作環境に移行しているプロダクションが多い．

以上のような，GPU のハードウェアとしての性能向上ばかりでなく，プログラマブルシェーダやシェーディング言語により，GPU の持つ機能をプログラムで利用できる環境も整備された．これらの GPU を取り巻く環境が劇的に進化したことにより，GPU をグラフィック処理専用の CPU の補助プロセッサとしてだけではなく，並列計算ユニットとして応用する研究が活発化している．

本論文では，はじめに GPU の環境整備について解説し，つづいて GPU コンピューティングで何が可能となり，今後どのような発展が期待できるかを，具体的な例を挙げながら述べる．

2. プログラマブルシェーダについて

最初に GPU という名称が使われた NVIDIA 社の GeForce 256（1999 年発表）では，頂点座標変換や頂点単位のライティングの計算を固定機能パイプラインで行っていた．現在の GPU のアーキテクチャにおいて，このパイプライン処理の工程をプログラミング可能にすることで，3DCG の高速で多様なエフェクトを演出することが可能になった．このパイプライン内のプログラミング可能な機能を，プログラマブルシェーダと呼ぶ．プログラマブルシェーダにより，従来はオフラインで行っていた様々なレンダリング手法がリアルタイムで実現でき，高品質な画像によるビデオゲームやインタラクティブシステムを制作することが可能になった．

2.1 3D グラフィックスの処理の流れ

はじめに，一般的な 3D グラフィックスの処理の流れを図１に示す．各段階の処理内容は以下のとおりである[1]．

図１　3D グラフィックスの処理の流れ

(1) シーンデータ作成：表示するモデルデータなどの作成やカメラおよびライトの設定．
(2) トラバーサル：シーンデータを解釈して必要な描画オブジェクトと描画命令を次段階へ渡す．
(3) ジオメトリ処理：モデルの座標変換、光源計算、投影変換、クリッピング処理などを行う．
(4) ラスタライズ：ジオメトリデータのラスタ化や各種シェーディング処理、Z バッファ処理、テクスチャ処理などを行う．
(5) 表示：ビデオメモリの値を読み出し，表示する．

以降述べるプログラマブルシェーダは，上記の(3)と(4)の処理の一部をカスタマイズするものである．

2.2 頂点シェーダとピクセルシェーダ

現在の GPU のパイプライン上でプログラミング可能な部分は，頂点単位での演算を行う Vertex Shader（Vertex Program とも呼ばれる．以下，頂点シェーダと称す）と，ピクセル単位での演算を行う Pixel Shader（Fragment Shader，もしくは Fragment Program とも呼ばれる．以下，ピクセルシェーダと称す）の２つがある．

頂点シェーダは，パイプラインを通る頂点データに対して，座標変換や頂点単位のライティングなどの計算を行うシェーダである．そして，頂点ごとのテクスチャ座標値や頂点法線などのデータを，ピクセルシェーダに渡す．

ピクセルシェーダでは，頂点シェーダの処理後，表示する各面上でのピクセル単位の処理を行う．例えば，テクスチャマッピングの手法や色づけの計算法などを記述する．また，ピクセルシェーダでは，フレームバッファへのデータ出力以外にも，深度バッファやステンシルバッファへのデータ出力が可能であり，影付けの事前処理などに用いられる場合もある．

2.3 シェーディング言語

GPU がプログラマブルなパイプラインを採用した当初は，プログラム言語としてアセンブラ言語が利用されていた．しかし，アセンブラ言語は Direct3D や OpenGL などの 3D API を用いている開発者には馴染みが浅いため，アセンブラ言語にかわる高級プログラム言語が策定されるようになった．

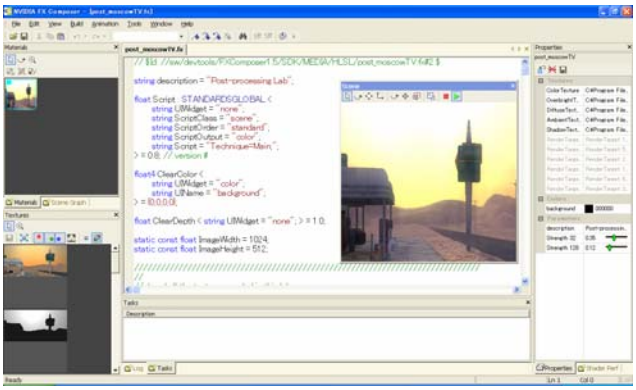
最初の GPU プログラミングの高級言語である C for Graphics（Cg）[2]の正式版が，NVIDIA 社から 2002 年にリリースされた．2003 年には，Microsoft 社が DirectX 9.0 のリリース時に，NVIDIA と開発協力をした Direct3D 用のシェーディング言語である，High Level Shader Language（HLSL）[3] を発表した．OpenGL でも，2003 年に発表されたバージョン 1.5 において，高級言語の OpenGL Shading Language（GLSL）[4]が標準仕様に盛り込まれた．

高級プログラム言語の策定には，GPU メーカや 3D API の標準化団体が主導している場合が多いが，大学などの研究機関からも，いくつかの提案がされている．例えば，Stanford 大学の Stanford Real-Time Programmable Shading Project[5]や Brook[6,7]，Waterloo 大学の Sh [8,9]などが挙げられる．Brook は GPU をグラフィック用のハードを意識させない汎用的な並列計算機（stream processor）として扱えるように C++をベースに設計されている．Sh では，3D API 側に標準化されたシェーディング言語と比較して，C++の STL[1]的な記法も取り入れた先端的な言語設計になっている．

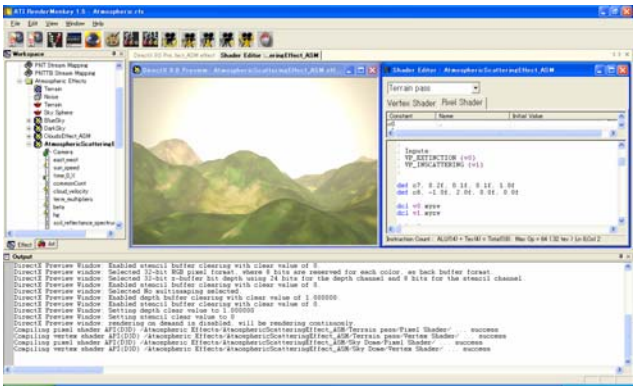
[1] C++でプログラミングを行なう際によく使う汎用的なデータ構造やアルゴリズムを、標準テンプレートライブラリとして利用しやすい形にまとめたもの

2.4 シェーダ開発環境

Cg, HLSL, GLSL などの主要なシェーディング言語は，3D API のプログラムコードとは独立して記述することができる．したがって，アプリケーション開発時に，プログラムを再構築せずにシェーダ機能の変更が適用できるため，開発の効率が向上する．



(a) FX Composer の操作画面



(b) Render Monkey の操作画面

図 2 シェーダ機能の統合開発環境の例

シェーダ機能のみを単独で開発できる統合開発環境が，GPU メーカなどから提供されている． 例えば，NVIDIA 社の FX Composer [10]，ATI 社の Render Monkey™[11]などが挙げられる．それぞれがプログラマに馴染みのある，エディタからコンパイラ，ビューワまでを備えたインタラクティブな統合開発環境となっている． FX Composer は HLSL のみをターゲットにし，コーディングが重視された作りになっている．一方，Render Monkey™ は，変数やリソースの利用および，マルチパスのシェーダ構成の編集などが GUI ベースで開発できる．図 2 にそれぞれの操作画面を示す．

3DCG ソフトウェアにおいても，シェーディング言語をサポートしているものが多い．例えば， Softimage XSI や 3D Studio MAX では，リアルタイムの 3DCG アプリケーション（主にコンピュータゲーム）の開発支援のために，Cg や HLSL をサポートしている．また，MAYA においては，NVIDIA 社が CgFX を用いたシェーダ機能の編集とレンダリング結果を閲覧する機能[12]を，Microsoft 社がエフェクトファイル形式を適用してのレンダリングを閲覧できる機能を，それぞれプラグインとして提供している．これらの機能は，シェーダ開発を支援するというよりも，アーティストがシェーディング効果を検証する意味合いが強いものとなっている．

2.5 シェーディング言語の将来

シェーディング言語は，3D API の挙動に関わる部分を制御できるように，さまざまな機能拡張がされている．すなわち，シェーダの実行時における，ハードウェア Z バッファなどの 3D グラフィックハードウェア機能の利用指定を，シェーダファイル内に埋め込むことができる．例えば，NVIDIA 社の CgFX，Microsoft 社では FX ファイルと呼ばれるメタファイルフォーマットが挙げられる．さらに DirectX 9.0 の 2004 年夏バージョンにおいては，アプリケーション間でのエフェクトファイルのシームレスな受け渡しができるようにするための記述方法として，DirectX Standard Annotations and Semantics (DXSAS) が提案されている．多くの場合，シェーダプログラムが 3D API 側から受け取る変数は，シェーダエフェクトの種類に関わらず同じ場合が多い．例えば，ライトやカメラなどの標準的なパラメータは多くのシェーダエフェクトで用いられる．したがって，開発者ごとにこれらの変数名が異なると，プログラムの可搬性が悪いものとなる．DXSAS では，標準セマンティクスを用いて変数やオブジェクトの共通の役割を記述し，標準アノテーションを用いて，さらに細かな動作を記述するようにされている．これにより，シェーダ内で利用する変数やオブジェクトの機能や役割を統一して記述することが可能になる．

一方で，モデルデータフォーマットとシェーダファイルとを結びつける提案も行われている．例えば，SCEA 社の XML 形式のモデルデータフォーマットである COLLADA[13]では，Cg 1.2 のサポートが仕様に組み込まれている．

プログラマブルシェーダの拡張として，Microsoft 社では，将来の Direct3D と言える Windows Graphics Foundation (WGF)において，現在の頂点シェーダとピクセルシェーダの間に，Geometry Shader（以下，ジオメトリシェーダと称す）と呼ばれる，新たなシェーダプログラミングのフェイズを導入することをアナウンスしている．ここで，ジオメトリシェーダは，頂点シェーダの処理後に形成される面に対して処理を行うための機能を持つ．

以上述べたように，シェーダ言語は，単にさまざまなシェーディングを可能にするためのプログラミング言語という位置づけから，3DCG アプリケーションのためのメタファイルフォーマットへと転換する過渡期にあると言えよう．

3. 拡大するGPUの応用範囲

一般的なGPUの価格は，PCを構成するパーツの中でも高価な位置づけにあり，中心的な役割を果たすパーツとなっている．従来は，CPUが汎用的な計算を行うためのユニットとして機能していた．しかし，2章で述べたように，GPUの持つ機能をプログラムで利用可能になったことにより，GPUはCPUに代わる並列計算可能な汎用かつ高速な計算ユニットとして活用されるようになってきた．

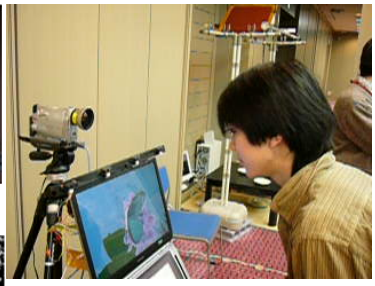
Eurographics2004のチュートリアルセッションにてNVIDIA社のHarrisは，3GHzのPentium4が理論上6GFLOPSの計算性能であるのに対して，GeFORCE FX 5900は実測20GFLOPS，GeFORCE 6800 Ultraでは実測40GFLOPSであると発表した．すなわち，計算ユニットとしてみても，CPUを凌駕するパフォーマンスを有することになる．GPUをグラフィックス以外の汎用目的に活用する研究は近年盛り上がりをみせている．例えば，2004年の8月には，ACM Workshop on General Purpose Computing on Graphics Processors (GP2) [14]が開催され，また，GPGPU [15]という名称のフォーラムも存在している．本章では，いくつかの具体的な例を挙げながら，GPUを汎用目的に活用した応用例を述べる．

3.1 画像処理

GPUのグラフィックス以外への応用としては，まず，ピクセルシェーダ機能を用いた画像処理の高速化が挙げられる．画像処理は画素単位で行うものが多く，実装上，ピクセルシェーダ機能の並列処理との整合性が高い．ガウシンアンフィルタやエンボス処理などの標準的な画像処理は，シェーディング言語の初歩的なプログラミングで実装可能である．

(a) 入力画像

(b) 画像処理結果



(c) インタラクティブシステムへの応用
(花にとまる蝶のCGコンテンツを鑑賞)

図3 GPUによる画像処理

筆者らは，すでに，動画像内の肌色領域の抽出処理をGPU上で行い，その高速化を検証した[16, 17]．この手法では，画像をGPUで処理可能なテクスチャデータに変換後，ピクセルシェーダ機能を用いて，RGB表色系から$L^*a^*b^*$表色系へ色空間を変換し，与えられた閾値で肌色領域を抽出している．この一連の処理をCPUおよびGPUで行い，その処理時間を計測した．Pentium 4 3.0GHz(HTテクノロジ対応)，RAM 1GB，RADEON 9800 XT 256MBを搭載したWindows PCで実験を行った結果，CPUの処理速度が38回/秒に対して，GPUは148回/秒と，4倍ほどの高速化が実現できることを確認した．図3に画像処理の例と，インタラクティブシステムへの応用例を示す．

一方，Yangらは，GPUを用いてステレオ画像からリアルタイムで深度値を計算する手法を開発した[18]．彼らは，異なる窓サイズに対するSSD法を用いた画像間のマッチング処理に対して，GPUのミップマップ機能を利用し高速化を行った．

また，Jahshaka [19]という名称のビデオ編集・ビデオエフェクト処理に関するオープンソースプロジェクトがある．このプロジェクトからは，GPUの機能を活かしたリアルタイムのビデオエフェクトツールが公開されている．

3.2 音声処理

リアルタイムでの音声処理は，DSPユニットを用いることが多いが，一般的にDSPをコントロールするAPIは公開されておらず，独自のサウンド効果を開発することは困難である．一方，GPUのピクセルシェーダ機能を応用することで，DSPにかわるリアルタイムでの音声処理をGPUでおこない，さまざまなサウンド効果の開発が可能になる．

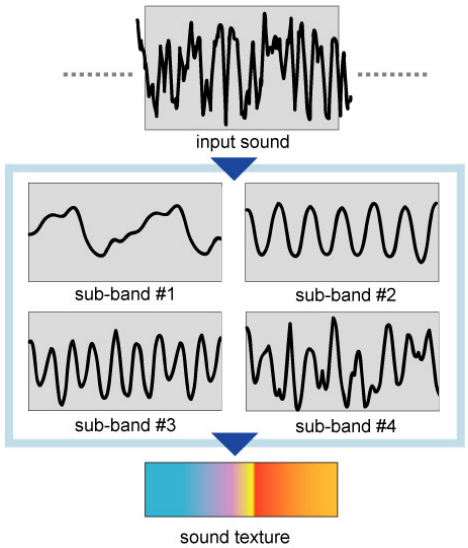


図4 GPUによる音声処理

GPUを用いた音声処理では，処理対象の音声データをテクスチャデータに変換して処理を行う場合が多い．図4に示すように，まず，入力音声を4帯域(例えば，-1000Hz，-2000Hz，-6000Hz，-22000Hz)に分ける．その後，各帯域で256階調に量子化し，テクスチャの4チャンネル(RGBA)に格納する．このように変換されたテクスチャデータに対して，例えば，テクスチャデータをスケーリングする

ことでドップラ効果を演出することができる.

BionicFX社[20]では, Audio Video Exchange (AVEX) と呼ばれる技術を用いて, デジタルのオーディオデータをグラフィックデータへと変換し, さまざまなオーディオ効果をグラフィックカード上で行うことを可能にした. 2005年の1月には, リバーブ効果 (残響効果) を演出するフィルタ処理をGPU上で行うVST[2]プラグインがアナウンスされる予定である. これにより, コンサートホールやスタジアムなどのさまざまな音場をGPUで再現することが可能になり, ミュージックプレーヤばかりでなく, カラオケのエコーや, ゲームなどへの適用も期待できる.

3.3 リアルタイムシミュレーション

3.3.1 流体解析

Harrisは, Stamの理論[21]に基づいた流体解析のリアルタイムシミュレーションをGPUで実装した[22]. この手法では, ナビエストークの方程式の解法にセミ・ラグランジェ法を応用し, 低精度ではあるが高速かつ安定に解いている. 図5にシミュレーション結果の一連の画像を示す. この例では, 512x512の画像サイズでGeForce FX 5950 Ultra (256MB)を用いており, 75FPS以上の描画速度でシミュレーションしている.

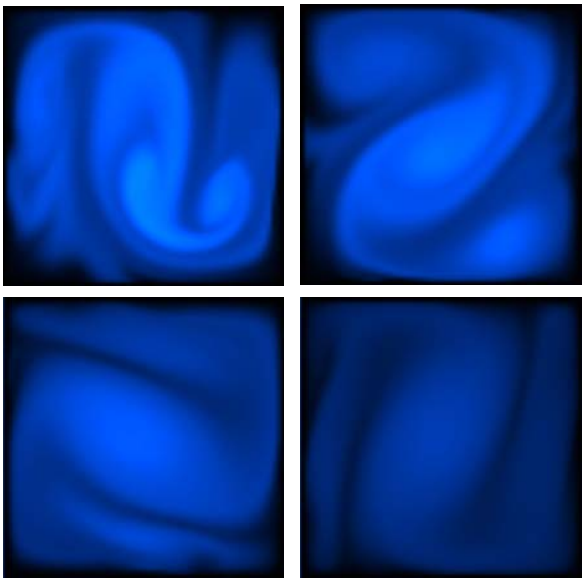


図5 流体のリアルタイムシミュレーション

3.3.2 形状変形のシミュレーション

Jamesらは, GPUを用いたリアルタイムでの形状の変形シミュレーション[23]を提案している. この手法では, 弾性体に対するモーダルな振動解析をあらかじめ行い, 各モードにおける頂点ごとの変形量をDynamic Response Texture(DyRT)と呼ぶデータとして求めておく. このDyRTを頂点シェーダで用い, 身体の動作に伴う胴や四肢の揺れ応答をリアルタイムで表現している.

3.3.3 ダイナミクスを応用したテクスチャ生成

Harrisらは, さまざまなダイナミクスに対してGPUを用いて高速にシミュレーションする手法を報告している[24]. この手法では, セルオートマトンを拡張したCoupled Map Lattice (CML)を用いて, 離散格子上のダイナミクスを計算している. この計算において, 各格子点をGPUのローカルメモリ内のピクセルに対応させ, 各ステップのシミュレーションをピクセルシェーダを用いて実装する. これにより, Reaction-Diffusionテクスチャや液体の沸騰などのビジュアルシミュレーションをリアルタイムで実現できることを報告している.

その他のGPUの応用例として, Govindarajuらは, GPUのパイプライン処理や, 並列処理, SIMD性能, ベクトル演算などの特性を活かし, データベース処理の高速化を行っている[25]. また, レイトレーシング[26]やラジオシティ法[27]などのレンダリング手法や, コンピュータビジョン[28]へのGPUの応用例も報告されている.

3.4 GPUの並列化

産総研のボリュームグラフィックス連携研究体は, 大規模ボリュームデータの計算と可視化を, リアルタイムで実行するPCクラスタベースのシステムを開発した[29]. このシステムはボリュームグラフィックスクラスタ (VGクラスタ) と呼ばれ, 並列化したGPUで生成される部分画像を, フレーム重畳装置で一枚の完成画像にする. VGクラスタを用いることで, 数値流体シミュレーションや３次元医用画像処理の高速な処理が可能となる.

一方Stony Brook 大学のFanらは, GPUクラスタによる高速な科学計算を行った[30]. 彼らは, GPUクラスタを構築して, Lattice Boltzmann Model (LBM) を用いた流体シミュレーションを並列処理し, ニューヨークのタイムズスクエアにおける大気汚染物質の飛散シミュレーションを行った. GPUを30台用いて, 480 x 400 x 80 のサイズのLBM を１ステップ0.31 秒で計算でき, 既存のPCクラスタの4.6倍程度の速度となることを確認した.

また, NVIDIA社のマルチGPU構成のアーキテクチャである SLI(Scalable Link Interface) [31]では, 複数のGPUを専用インタフェイスで接続し, 処理の負荷分散が行える. SLI はシリアル転送インタフェイスである PCI Expressに対応した技術である. PCI Express x16の最大転送速度は4GB/秒であるため, 現在のグラフィックカードの接続規格であるAGP 8xの約2倍の転送能力を持つこ

[2] VSTとはVirtual Studio Technologyの略で, Steinberg社(ドイツ)が提唱するDTMのコンセプトである. この規格に対応したプラグインを用いることで, さまざまなエフェクトやミキシングなどが行える.

とになり，より高速なグラフィックデータの転送が可能になる．SLI 対応の GPU を用いることで，低価格での GPU サーバを構築することも可能である．

4. ユーザインターフェイスの変貌

現在のコンピュータのユーザインターフェイスには，大きく２つにわけて，文字ベースの CUI(Character User Interface)と，グラフィックベースの GUI(Graphical User Interface)がある．初期のパーソナルコンピュータや UNIX には，前者の CUI が主に用いられてきた．その後，1984 年に登場した Macintosh の GUI の成功により，ユーザインターフェイスの GUI 化が加速した．これに伴い，グラフィックを画面に高速に描画するアクセラレータ機能を持つグラフィックチップが多数登場することになる．

4.1 GUI とハードウェアアクセラレーション

グラフィックスのアクセラレータ機能がない場合，直線を画面上に１本描画するだけでも描画すべきすべての点を CPU で計算し，その結果をすべてビデオメモリに転送しなければならない．したがって，システムバスや CPU への負担は多大なものとなり，スムーズな GUI を実現することは困難になる．

現在の 2D グラフィックスのアクセラレータ機能としては，直線や矩形の描画や，描画する矩形領域外の非表示処理であるクリッピング処理，画面の矩形領域内の画素データ転送機能である BitBlt，ハードウェアカーソル機能などが挙げられる[1]．これらの機能をハードウェアレベルで提供することにより，CPU に負荷をかけずスムーズなコンピュータの操作が可能になっている．

4.2 GPU を活用したユーザインターフェイスへ

Macintosh の誕生から 20 年経ち，ハードウェアの性能が飛躍的に進化したのに対し，GUI の基本的なコンセプトはほとんど変化しておらず，その構成要素は従前として WIMP (Window, Icon, Menu, Pointer) のままである．

現在主流となっている GUI は，現実世界のデスクトップのメタファを 2D のグラフィックスで表現している．GPU を活用して表現を単に 3D 化し，現実世界をよりリアルに模しただけでは，ユーザビリティの観点からユーザインターフェイスを改善するものとは思えない．

SUN 社の”Project Looking Glass” [32]では，JAVA™ 技術に基づいた 3 次元ウィンドウ環境の構築を進めている．Project Looking Glass では，3 次元空間を利用した奥行きや透明度，傾きの表現を伴うウィンドウ操作により，アプリケーションやコンテンツを迅速かつ効率的に管理することを目指している．

一方 Microsoft 社では，次期 OS の“Longhorn”[33]でのプレゼンテーションサブシステムである“Avalon”において，GPU を有効利用した Direct3D ベースのユーザインターフェイスの構築を試みている．Direct3D ベースに GUI が構築されることで，従来の 2D の GUI と 3DCG や動画などのメディアが統合される．また，滑らかなテキストを表示する“ClearType”の処理速度が向上し，ウィンドウを任意の DPI にスケーリングしたり，その透過率を変更して表示させることも可能になる．しかし，Avalon の最大の特徴は，XML ベースの XAML(eXtensible Application Markup Language)を用いて，アプリケーションのビジュアルインターフェイスをカスタマイズできることにある．すなわち，ビジュアルインターフェイスを XAML で定義し，コアとなる処理には従来のプログラムコードで記述する，というアプリケーションの構築を可能にする．これにより，アプリケーションのユーザインターフェイスデザインがより簡単になり，マークアップ言語による WEB プログラミングモデルと，コンパイラを中心とした従来のプログラミングモデルが収束に向かうとされている．

4.3 携帯端末の GPU の発展

従来，携帯端末の埋め込み GPU は，2D のグラフィック描画や，MPEG/JPEG のデコーダ機能を受け持つ統合メディアプロセッサとしての位置づけであった．しかし，昨今の携帯端末上で動作するコンテンツにリアルタイムでの 3D グラフィックスが多用されるようになり，PC 並みの描画処理能力が要求されるようになった．

ATI 社からは IMAGEON™2300，NVIDIA 社からは GoForce3D 4500 が既に発表されている．また，BitBoys 社[34]からは，G40 などが発表されている．これらの GPU に対しては，OpenGL ES(Embedded Systems)，Direct3D Mobile，M3G (JAVA の 3D API，mobile 3D graphics API for J2ME)などの API が用いられる．

携帯端末，特に携帯電話における 3D グラフィックの利用目的は，ビデオゲームばかりでなく，マクロメディア社の Shockwave を利用したメディアリッチな WEB サイトの閲覧など，ユーザインターフェイスの高品質化が，今後一層重要となるであろう．

5. まとめ

現在，携帯電話から，PDA，ビデオゲーム機や PC に至るあらゆる情報端末に GPU が搭載されている．この動きは，ユーザが高品質なグラフィックスを体験できるようになったばかりではなく，コンテンツの制作側から見ると，グラフィックコンテンツの実行環境が真の意味でクロスプラットホームになりつつあると言える．今後は，ハイエンドからローエンドにいたる環境まで，トップダウンにコンテンツの流用が可能になり，制作効率がさらに向上するであろう．

従来の CPU 中心の計算環境から GPU 中心の計算環境への

移行は予想以上に早く進んでおり，PC における GPU の位置づけは極めて重要なものになりつつある．GPU の汎用並列計算機としての利用を考えた場合，グラフィックス独特のアーキテクチャを理解せずに扱えるように改善されていくことが予想される．例えば，配列型データは，テクスチャの各ピクセルを配列要素として扱っており，グラフィックスのプログラミングに馴染みの浅い人には一般的ではない．さらに，精度面では，浮動小数点演算が単精度から倍精度へと引き上げられる必要がある．GPU のハードウェア性能と環境がここまで整ったときに，初めてスーパーコンピュータの代替品としての要求を満たすのではないかと考える．

GPU のプログラミングモデルは，汎用並列計算機としての，より柔軟性の高い仕様になると予測される．ただし，現状の GPU プログラミングでは，GPU が固定機能で実装していた低レベルの部分をプログラミングする必要がある．例えば，頂点座標変換は 3D API で用意された処理であるが，GPU では実装が必要である．したがって，こうした手間を代替するミドルウェアやライブラリの整備も急務であろう．

今後，CPU と GPU を融合した強力な計算能力を持つハードウェアが安価かつ容易に扱える時代を迎えるにあたり，GPU コンピューティングを支える人材の育成が，いっそう重要になるものと考える．

参考文献

[1] 宮田一乗, “PC グラフィックボードの現状と動向”, 映像情報メディア学会誌, Vol.55, No.8/9, pp.1094-1099, 2001.

[2] R. Fernando, and M. J. Kilgard, “The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics,” Addison-Wesley, 2003.

[3] Kris Gray, “プログラミング Microsoft DirectX 9 グラフィックス パイプライン”, 日経 BP ソフトプレス, 2004

[4] R. J. Rost, J. M. Kessenich, B. Lichtenbelt, M. Olano, ″ OpenGL Shading Language,″ Addison-Wesley, 2004.

[5] リアルタイムシェーディングプロジェクトのサイト, http://graphics.stanford.edu/projects/shading/

[6] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan,, “Brook for GPUs: Stream Computing on Graphics Hardware,” ACM TOG, Vol.23, No.3, 777-786, 2004.

[7] Brook のサイト, http://brook.sourceforge.net

[8] Sh のサイト, http://libsh.org/

[9] M. McCool, and S. DuToit, ″Metaprogramming GPUs with Sh,″ A K Peters Ltd., 2004.

[10] FX Composer のサイト, http://developer.nvidia.com/object/fx_composer_home.html

[11] RenderMonkey のサイト, http://www.ati.com/developer/rendermonkey/index.html

[12] CgFX プラグインのサイト, http://developer.nvidia.com/object/cgfx_mel.html

[13] Collada のサイト, https://collada.org/

[14] GP2 研究会のサイト, http://www.cs.unc.edu/Events/Conferences/GP2/index.shtml

[15] GPGPU フォーラムのサイト, http://www.gpgpu.org/

[16] 高橋, 河原塚, 宮田, “GPU による肌色認識処理の高速化に関する一手法”, 第3回NICOGRAPH 春季大会論文集, pp.55-56, 2004.

[17] 河原塚, 高橋, 宮田, “ViewFrame2 －マーカレス顔部検出手法を利用した“ViewFrame”－”, 芸術科学会論文誌, Vol.3, No.3, pp. 189-192, 2004.

[18] R.Yang and M.Pollefeys, ″Multi-Resolution Real-Time Stereo on Commodity Graphics Hardware,″ CVPR 2003, pp.211-217, 2003.

[19] Jahshaka のサイト, http://www.jahshaka.com/

[20] BionicFX 社のサイト, http://www.bionicfx.com/

[21] Jos Stam, “Stable Fluids,” In Proc. of SIGGRAPH’99, pp.121-128, 1999.

[22] Mark Harris, “Fast Fluid Dynamics Simulation on the GPU”, pp.637-665, in Randima Fernando, “GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics,” Addison-Wesley Pub., 2004.

[23] D. L. James and D. K. Pai, ″DyRT: Dynamic Response Textures for Real Time Deformation Simulation with Graphics Hardware, ″ACM TOG, 21(3), pp. 582-585, 2002.

[24] M.J. Harris, el.al., “Physically-Based Visual Simulation on Graphics Hardware,” In Proc. of Graphics Hardware 2002, pp.1-10, 2002.

[25] N. Govindaraju, et al., “Fast Database Operations using Graphics Processors,” Proc of ACM SIGMOD 2004, pp.215 - 226, 2004.

[26] T.J. Purcell, I. Buck, W. R. Mark, P. Hanrahan, ″Ray Tracing on Programmable Graphics Hardware, ″ ACM TOG, Vol.21, No.3, pp. 703-712, 2002.

[27] G. Coombe, M. J. Harris, A. Lastra, ″Radiosity on graphics hardware,″ ACM Proc. of the 2004 conference on Graphics Interface, pp.161-168, 2004..

[28] J. Fung, and S. Mann, ″Computer Vision Signal Processing on Graphics Processing Units″, Proc. of the IEEE ICASSP 2004, Vol.5, pp.93-96, 2004.

[29] S. Muraki, et.al., ″Next-Generation Visual Supercomputing using PC Clusters with Volume Graphics Hardware Devices,″ Proc. IEEE SC2001, 51, 2001.

[30] Zhe Fan, et.al., “ GPU Cluster for High Performance Computing,” to appear in ACM/IEEE SuperComputing 2004 (SC’04), November, 2004.

[31] NVIDIA社SLIのサイト, http://www.nvidia.com/page/sli.html

[32] Project Looking Glass のサイト, http://wwws.sun.com/software/looking_glass/

[33] Longhorn のサイト, http://msdn.microsoft.com/longhorn/

[34] BitBoys 社のサイト, http://www.bitboys.com/