PHIGS のジオメトリ演算のための並列処理方式の検討

松本 尚[†] 川瀬 桂[††] 森山孝男[††]

ポリゴンベースのレンダリングを行う三次元グラフィックスシステムにおいて，描画速度の高速化と表示画像の高品位化に伴って，ジオメトリ演算部の処理能力が高性能化のボトルネックになるようになった．このボトルネックを解消するために，ジオメトリ演算部に並列処理の導入が図られるようになってきている．また，今後開発されるグラフィックスシステムはグラフィックスインタフェースとして標準の地位を確立しつつあるPHIGS の採用が不可欠である．本論文では，PHIGS を採用した高性能三次元グラフィックスシステムの構築を可能にするジオメトリ演算部の並列処理方式について検討を行う．まず，プロセッサの使用効率などの面から均質型マルチプロセッサがパイプライン構造の並列処理より優れていることを定性的に示す．次に，定量的な検討を加えるためにプリミティブ単位のジオメトリ演算部内での処理量とデータ転送量を見積もり，さらにシステム全体の目標性能とジオメトリ演算部に要求される性能の関係を評価する．その結果，基本的にジオメトリ演算がコンピュテーションインテンシブであるため内部のデータ転送路に構造の簡単なバスを使用できることを示し，プログラミングモデルの単純さなどから共有メモリ型の優越性を述べる．そして，アーキテクチャとして共有メモリ共有バス型マルチプロセッサを採用する場合に，PHIGS 特有のデータ構造から派生するプロセッサごとの TSL (Traversal State List) の管理の問題を解決する処理方式およびそのハードウェア支援機構を提案する．

A Study of Parallel Processing Methods for Geometric Calculation in PHIGS

Takashi Matsumoto,[†] Kei Kawase[††] and Takao Moriyama[††]

High-speed 3D graphics is a key technology for developing more user-friendly man-machine interfaces. In present high-end graphics systems, geometric calculations for 3D graphic images are bottlenecks in system performance. To solve this problem, parallel processing technique is being introduced. If the Programmer's Hierarchical Interactive Graphics System (PHIGS) is adopted as an application programmer's interface, however, there are some obstacles to efficient parallel processing. In the first part of this paper we show that homogeneous multiprocessors are superior to pipeline-type multiprocessors in geometric calculations. To obtain quantitative data for the design of a geometry subsystem, we analyze the characteristics of geometric calculations (coordinate transformations, lighting calculations, clipping calculations, and data translations). From the analysis we conclude that a simple shared-bus architecture can be used for a geometry subsystem, and that a shared-memory architecture is superior to a distributed-memory one in terms of generality. To solve difficulties resulting from the adoption of PHIGS, we finally propose two mechanisms that enable shared-memory/shared-bus multiprocessors to perform efficient geometric calculations in PHIGS.

1. はじめに

本稿では**図1**のような構成を持つ高速な三次元グラフィックスシステム（ポリゴンレンダリングベース）を仮定し，ジオメトリ演算部における並列処理方式について議論を行う．

従来より，ドローイング部はスキャンライン単位やピクセル単位の処理の並列性を用いた高速化がなされ，1秒間に100万個以上のポリゴンをラスタライズする能力を持つものも開発されている[1)-4)]．そして，近年ではポリゴンベースのグラフィックスシステムも高速性だけではなく画像の品質も求められるようになってきている．その実例として，最近の多くの商用機ではテクスチャマッピングやアンチエイリアシングのためのハードウェア支援機構が用意されている．ま

† 東京大学理学部情報科学科
Department of Information Science, Faculty of Science, University of Tokyo
†† 日本アイ・ビー・エム(株)東京基礎研究所
Tokyo Research Laboratory, IBM Japan

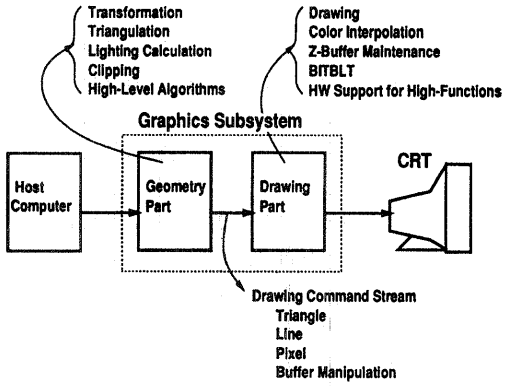


図 1 システムの全体構成
Fig. 1 An overview of the system.

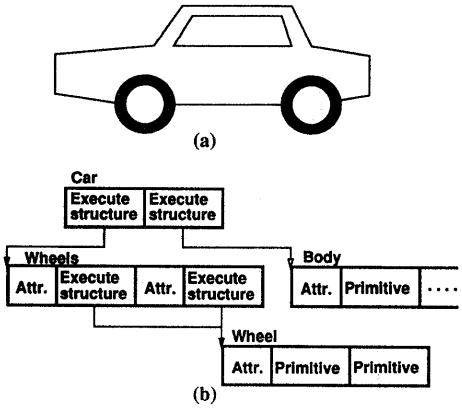


図 2 階層状データ構造の例
Fig. 2 An example of hierarchical data structure.

た，メモリの大容量化と低価格化の恩恵にあずかって開発されたアキュームレーションバッファ[5]を用いれば，今までポリゴンベースのレンダリングでは難しいとされていた高品質な画像の作成も可能になった．しかし，高品位画像を得るためには，複数の光源を用いたり，曲面をアダプティブに細かく分割したり，複数の画像を生成してそれらを1枚に合成するといった処理が必要とされ，ジオメトリ演算部において従来以上に大量の演算が必要とされるようになってきた．そこで，並列処理を利用したジオメトリ演算部の高速化が不可欠となってきている．

一方，グラフィックスインタフェースとしてPHIGS (Programmer's Hierarchical Interactive Graphics System)[6),7)]が標準の地位を確立しつつあり，今後開発されるグラフィックスシステムはPHIGSのサポートを避けて通れない．PHIGSはモデリング時のデータの定義編集の容易さと即時性，さらに大規模データ取り扱い時のメモリ効率を考慮して，階層状データ構造を採用している．このデータ構造で記述された物体を画面に表示する際には，階層状データ構造の中を順にたどってデータを展開する必要がある．この展開の処理（トラバーサル：Traversal）が本質的に並列性のない処理であるため，必ずしも並列処理と相性が良くない．そこで，PHIGSの階層状データ構造を処理することを考慮して，ジオメトリ演算部に適した並列処理方式およびハードウェア構成を模索する．

2. PHIGS のデータ構造とトラバーサル

まず本章ではPHIGSの階層状データ構造とそのトラバーサルについて簡単に解説する．

PHIGSではモデリング・データを階層状のデータ構造として記憶域に格納している．今後，この格納場所を**CSS** (Central Structure Store) と呼ぶ．このデータ構造はさらに**構造体** (Structure) と呼ばれる下部構造を持つ．構造体はプログラミング言語のサブルーチン・コールのように他の構造体を呼び出すことができる．この呼び出しによりモデリング・データが階層状のデータ構造を形成する．各構造体は**構造体要素**から構成され，構造体要素にはこの構造体呼び出しのほかに，各種の**図形要素**（**プリミティブ**：点，直線，ポリゴン等）の描画を指示するものと**属性**（**アトリビュート**：プリミティブの色，光の拡散率，変換マトリックス等）の設定を指示するものがある．例えば，**図2**（a）のような形状（便宜上二次元）を構成する場合，図2（b）のように構造体 Car が構造体 Body と構造体 Wheels を呼び出し，さらに構造体 Wheels が構造体 Wheel を呼び出すことで階層状にデータが構成できる．（図2中の Attr. は属性設定を示す．）

モデリング・データを表示する際は，CSS内の階層状に構築されたデータを構造体呼び出しのチェインをたどって「**順次列**」に展開する（これを**トラバーサル**と呼ぶ）．トラバーサルの結果は，例えば，**図3**のような図形要素の出力と属性設定を指示する構造体要素の列になる．この要素を順に処理することで画像を生成する．トラバーサルの処理自体（単なるポインタ追跡）には順次性があり，並列処理による高速化ができない．しかし，トラバーサルの処理とトラバーサル中に次々と生成されるプリミティブの描画要求は並列処理可能である．

属性（プリミティブの色，光の拡散率，変換マトリ

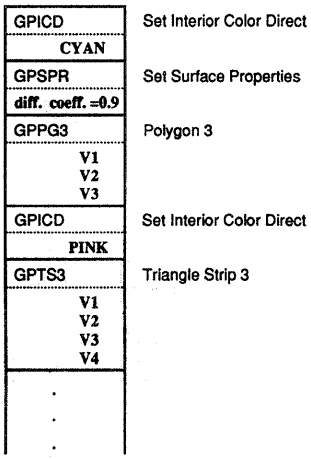


図 3 トラバーサル結果の順次列の例
Fig. 3 An example of display list generated by structure traversal.

ックス等）に関して少し説明を追加する．属性はトラバーサル結果の順次列中の属性設定の指令によって刻刻と変化し，ステートマシンを構成している（ある属性に対して設定された値は再設定されるまで有効）．今後，このステートの格納場所を **TSL** (Traversal State List) と呼ぶ．プリミティブの描画の指示は，その描画の指示が順次列で出現した時点での TSL 内の属性値を参照しながら処理されることが要求される．

3. パイプライン構造の欠点

従来のシステムもジオメトリ演算部に並列処理を導入していた．典型的な例としては，パイプライン状にプロセッサや演算器を配置して，レンダリングに必要なジオメトリ演算を分割し，パイプラインの各ステージに割り当てたものである[8]．例えば，最初のプロセッサは座標変換，二番目はライティング計算，三番目はクリッピング計算，四番目は透視変換のための割算，五番目はデプスキューイングやドローイング部へのデータの加工（浮動小数点数から固定小数点数への変換等）を行うといったように役割分担がなされる．このパイプライン構造はハードウェアが比較的単純，データの処理順序が入力と出力で保存される，プリミティブ単位に必要とされる一連の処理のソフトウェア側からの描像と構成がナイーブに対応がとれるといった利点がある．しかし，その反面以下のような欠点を持っている．

1. **演算器間（ステージ間）の負荷分散が難しい．** パイプラインの各ステージの役割分担を決める際に，なるべく負荷の大きさが揃うように設計を行ったとしても，データによって実行時の計算量が大幅に変わるため，完璧な静的負荷分散は不可能である．例えば，ライティング計算では考慮すべき光源の種類や数によって，計算量が大幅に変化し，またクリッピングでは図形が分割されない場合と分割される場合で大きく計算量が異なる．このため，ステージ間の負荷にアンバランスが生じ隘路が発生し，実用上は演算器の使用効率が上がらず，性能の向上が難しい．
2. **ステージ間のデータの移動量が多く，局所性が利用できない．** 一連の処理を分割して異なるステージで処理しているため，前段のステージから入力データを受け取って次段のステージに出力する間の処理量は，単一のプロセッサで一連の処理をまとめて行う場合に比べて小さい．このため，利用できる局所性が小さく，データのステージ間の移動に伴う入出力のオーバヘッドも無視できない．
3. **ハードウェアの構造とプログラム（マイクロコード）が密に関連し，スケーラビリティやフレキシビリティが乏しい．** パイプライン構造では他のステージの計算の途中結果を用いるアルゴリズムやリカーシブなアルゴリズムを実装できない．また，パイプラインのステージ数等の構成が変わるとプログラムの大幅な書き換えが必要になり，ステージ数の増加による処理能力のスケールアップは難しい．

上記のような理由により，パイプライン構成以外の並列アーキテクチャが高速なジオメトリ演算部に望まれる．

4. 均質型マルチプロセッサの利点

最近，強力な浮動小数点演算器を内蔵したマイクロプロセッサの出現に伴い，ジオメトリ演算部にこのタイプのマイクロプロセッサを複数搭載した均質型マルチプロセッサを採用するシステムが現れた[9),10)]．均質型マルチプロセッサを採用する場合，パイプライン構造の欠点として掲げた点を克服することができる．

負荷分散に関しては，一つのプリミティブ（ポリゴンやライン）の表示のための処理をパイプライン状に分割するのではなく，手の空いた一つのプロセッサに一つのプリミティブの表示のためのジオメトリ計算をひとまとめにして順に割り当てる**順次振り分け方式**を

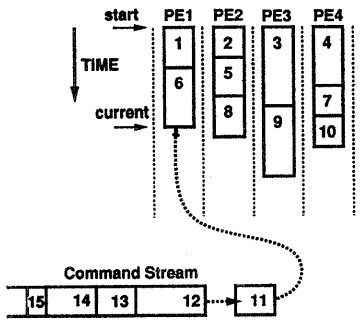


図 4 順次振り分け方式の負荷分散
Fig. 4 Load balancing by per primitive dispatch.

採用している[9),11)]．この様子を図解すると図4のようになる．図中でボックスは1個のプリミティブに対する計算を表し，コマンドストリームは構造体をトラバースした結果のプリミティブ表示の処理要求列を示し，ボックスの長さは処理の大きさ（計算時間）を，ボックス内の数字はトラバーサルの順次性から派生する順序を表している．シェーディングが施されたポリゴン表示のためのジオメトリ演算の計算量（概算については後述）はかなり大きく，順次振り分けは処理が単純であるため，負荷分散のためのオーバヘッドは相対的に小さい．さらに，通常ポリゴンベースのグラフィックスシステムではデプスバッファによる隠面消去をドローイング部内で行うので，基本的に各プリミティブを独立に処理することができる．そこで，異なるプリミティブを同時に複数の要素プロセッサで並列処理することによりジオメトリ演算が高速化できる．この場合，動的に負荷を分散することができるので，要素プロセッサへのデータ供給がボトルネックにならなければ，要素プロセッサの使用効率を100%近くにすることができる．

過去のアプリケーションにはトラバーサルの順次性を積極的に使用しているものがあり，トラバースされた順序でフレームバッファにプリミティブを描画することを要求する．これらのアプリケーションに対してはジオメトリ演算部からドローイング部へのデータの出力に関してトラバーサル時の順序（つまり，ジオメトリ演算時のデータの入力順序）を保存する必要がある．この制約は，ジオメトリ演算部にデータ出力をバッファリングする能力（一時的にデータを保持する記憶領域）があれば，順序を整えて出力する処理のオーバヘッドは生じるが，要素プロセッサの使用効率を落とさずに解決できる．ただし，デプスバッファ法は本来プリミティブの描画順序に制約を加えないことを特長としており，以下の議論ではプリミティブのフレームバッファへの描画順序に制約がないものと仮定する．

また，順次振り分け方式の場合は，プログラムを変更することなしに，要素プロセッサの台数を変更することで処理性能を増減させることも可能になる（スケーラブルな構成にできる）．このため，能力別に複数の製品モデルを開発する際に，大きなメリットがある．

ただし，この順次振り分け方式では，TSLのスナップショット（ある時点における TSL の内容）がプロセッサごとに保持される必要がある．なぜなら，共有メモリ上に全プロセッサで共通の TSL を設けると，あるプロセッサがプリミティブの描画を行っている間は TSL はそのプリミティブのためのスナップショットに固定しておく必要があり，そのプリミティブの描画の指示に続く属性設定の指示を実行することができない．このため，構造体要素の順次列の処理を先に進めることができず，プリミティブ単位の並列実行ができない．

5. プリミティブ単位の計算量およびデータ転送量の見積り

定量的な議論の基礎データとして，三次元のシェーディング付きの画像生成において使用頻度の高い3種類のプリミティブを対象にして，ジオメトリ演算部における計算量とデータ転送量の概算を求めた．

まず，この概算のための仮定について述べる．ジオメトリ部の演算器として LIW (Long Instruction Word) 型[12)]またはスーパスカラ型のマイクロプロセッサを用いるものとし，このプロセッサは単体で浮動小数点加算および乗算，整数演算，メモリアクセスの四操作を同時に実行できると仮定する．演算はレジスタベースで浮動小数点演算は3クロック，整数演算は1クロックのコストがかかる．ただし，浮動小数点演算はパイプライン実行が可能で1クロックごとに演算をパイプラインに投入できるとする．また，割算と平方根の逆数を高速に計算するために，ニュートン法を実行するための種を求める命令（1クロックのコスト）が用意されているものとする．プロセッサは十分な容量の命令キャッシュを持ち，命令フェッチのコストはない．さらに，プロセッサはデータキャッシュを持ち，前出のプロセッサローカルの TSL や NURBS

曲面のポリゴンへの展開時の中間データ等はキャッシュされているものとする．レジスタ・メモリ間のデータ転送命令のコストは本概算ではすべて1クロックと仮定し，外部バスアクセスに伴う時間的オーバヘッド（レイテンシ）を無視する．

次に，概算の算出方法について述べる．ジオメトリ演算部で行われる処理をC言語でプログラムとして記述し，そのプログラムをハンドコンパイルすることで計算量（クロック数）の評価を行った．行列の乗算や割算といった定型処理の部分は完全にコードが最適化されている．プログラムには座標変換，ライティング計算，クリッピング，ドローイング部へのデータの出力処理が含まれる．ドローイング部は三頂点のRGBの輝度値とスクリーン座標とデプス値を受けとってデプスバッファ法と輝度の線形補間（グーローシェーディング）を行いながらフレームバッファ上に三角形を描くものとする．

以上の仮定と方法に基づいて3種類のプリミティブ（Polygon3, Triangle Strip, NURBS Surface）について処理時間とデータ転送量（プロセッサへの外部からの新規入力データのための転送量）の評価を行った．ただし，すべての評価はドローイング部への出力三角形1個当たりの処理時間および転送量であり，透視変換を含み，クリッピングによって分割されない（クリッピング処理はチェックのみ）と仮定する．また，座標データや法線ベクタとして単精度浮動小数点数（4バイト長）を用いている．

Polygon3 元データは三角形（三頂点のみを含むPolygon3），照度計算はポリゴン当たり1回（Per Area Lighting），点光源が3個という仮定の下で，**569** クロック．プロセッサの新規入力は三頂点の座標データで**36**バイトのデータ量．

Triangle Strip 照度計算は頂点当たり1回（Per Vertex Lighting）．処理時間は光源の種類と数によって**表1**のようになる．プロセッサの新規入力は一頂点の座標と法線データで **24** バイトのデータ量．

表 1 Triangle Strip に対する処理時間
Table 1 Processing time for Triangle Strip.

Lights		Clocks
Positional	Directional	
1	0	339
3	0	469
3	3	635

表 2 NURBS 曲面に対する処理時間
Table 2 Processing time for NURBS.

Lights		Clocks
Positional	Directional	
1	0	236
3	0	301
3	3	384

NURBS Surface NURBS 曲面は前もって，オスロアルゴリズム[13]等を用いてベジエ曲面に変換してから取り扱われる．曲面は次数が三次でトリムされておらず，固定ピッチで四辺形のポリゴンに細分割され，この四辺形の頂点ごとに照度計算が行われる．さらに半分の三角形としてドローイング部に出力される．出力三角形1個当たりの処理時間（処理の最内側ループのみの時間）は光源の種類と数によって以下の**表2**のようになる．

NURBS曲面の場合，プロセッサがドローイング部に出力する三角形を生成する最内側ループを回っている間は，新規のデータを外部から取り込む必要はない．分割されたポリゴンの生成前の前処理段階でコントロールポイントやノット（knot）ベクタを読み込む．10ポイント四方のコントロールポイントから定義されるNURBS曲面をパッチごとに100個の四辺形に近似して表示した場合，データ入力量は出力三角形一つ当たり **0.18** バイトになる．

プロセッサからドローイング部への出力のためのデータ量は，三角形1個をデプスバッファ法を行いながら，スクリーンにシェーディング付きで描画するのに必要なデータ量で，プリミティブの種類に依らない．ジオメトリ演算部で輝度やデプス値のX軸方向やY軸方向の傾斜まで求めるかどうかによって大幅に転送量が異なる．今回の評価ではこれらの傾斜はドローイング部で頂点のデータから求めることを仮定している．この仮定の下で，プロセッサからの出力データ量は**48**バイト程度（これにはポリゴンのIDや属性等も若干含まれており，純粋に三角形の表示のみならば36バイト程度）である．なお，処理時間（クロック数）の内訳を付録の表4，5，6に添付する．

6. ジオメトリ演算部に要求される処理能力の見積り

本章ではシステム全体の表示能力の目標性能とジオ

メトリ演算部に要求される処理能力の関係について，前章で示した出力三角形当たりの処理量とデータ転送量を基に議論を行う．

システム全体の表示能力を示す場合，Triangle Strip を単一点光源でライティング計算を行いながら表示したときに1秒間で表示できる三角形の数を用いることがある．表示能力に関して，本章ではこの指標を採用することにする．ジオメトリ演算部内のプロセッサは前章で仮定したスペックのものを用い，クロックの周波数は 50 MHz と仮定する．以下に，0.5 M, 1M, 2.5 M, 5M Polygon/sec の表示能力を得るのに必要なジオメトリ演算部内のプロセッサ台数，CSS と要素プロセッサ間のデータ転送能力，およびジオメトリ演算部とドローイング部間のデータ転送能力を**表3**にまとめる．ただし，順次振り分けやトラバーサルのオーバヘッドは無視できるとする．

CSS と要素プロセッサとの間の転送量については，データにヘッダが付加されたり，属性設定の情報が転送されることにより実際には表の値より若干大きな転送能力が要求される．

表3内のデータ転送量の値は現在のテクノロジを使用しても短い転送距離であれば一本のバス（64 bit 幅程度）によって十分実現可能な値である．しかし，現実的にはホストとグラフィックス・サブシステムの間の結合は転送能力の低い規格化されたバスやシリアルリンクがホストコンピュータの都合で採用される場合がある．その場合は，表示時にこの転送能力がネックになることを避けるために，CSS をジオメトリ演算部内に設け，構造体のトラバーサルをジオメトリ演算部内のプロセッサで行う必要がある．CSS がジオメトリ演算部内にある場合は表示時にホストとの間でデータは転送されず，構造体データの追加更新時のみデータが転送される．この構造体データの操作は人間との間の対話的操作である場合が多く，データ転送能力はあまり問題にならない．以下，CSS はジオメトリ演算部内にあると仮定する．

表 3 ジオメトリ演算部に要求される仕様
Table 3 Requirement for geometric calculation part.

Triangles (Poly/s)	CPUs (#CPU)	from CSS (Byte/s)	to Drawing (Byte/s)
0.5M	4(3.4)	12M	24M
1M	7(6.8)	24M	48M
2.5M	17	60M	120M
5M	34	120M	240M

7. ジオメトリ演算部の内部構造

本章では，ジオメトリ演算部内のマルチプロセッサの構造について検討を行う．順次振り分け方式で負荷分散を行うので，あるプリミティブのジオメトリ演算をプロセッサが計算している間は，他のプロセッサと通信する必要がない．よって，TSL のメンテナンスのためのデータの移動（この点は次章において議論される）を除けば，演算部内の基本的なデータの転送量は表3のジオメトリ演算部と外部との間の2種類のデータ転送量（表3の3列目と4列目）の和である．この値は目標性能が 5 M Polygon/sec であっても 360 M Byte/sec 程度であり，ジオメトリ演算部内にはあまり複雑な通信網を必要としない．そこで，ジオメトリ演算部内のデータ通信路として実装の容易なバス方式が採用可能である．

メモリ管理の容易さとプログラミングの容易さから，ジオメトリ演算部内の CSS は，すべてのプロセッサから同等にアクセスできる大容量の共有メモリにあることが望ましい．また，共有メモリベースのマルチプロセッサであれば，プログラミングモデルが単純で，マイクロコードの短時間での開発が期待できる．つまり，負荷分散の部分のコードは並列処理を意識する必要があるが，他の部分は単一プロセッサの場合と完全に同じになる．さらに，共有メモリベースのマルチプロセッサを採用する利点として次のような点も挙げられる．ジオメトリ演算部を汎用のアクセラレータとして流用する場合や，グラフィックスにおいてもレイトレーシングやラジオシティの計算を行わせるような場合に，単一プロセッサ用のプログラムの移植や並列化が簡単であり，さまざまなアルゴリズムを効率良く実現することができる．

以上のような考察の結果，**共有メモリ共有バス型マルチプロセッサ**がジオメトリ演算部の並列アーキテクチャに適していると考えられる．現状のテクノロジで，表3の目標性能が 2.5 M Polygon/sec ぐらいまでであれば，64 bit 幅程度のシングルバスをすべてのデータ転送路として用いることができる（**図 5**）．また，ジオメトリ演算部からドローイング部へのデータ転送は同じサブシステム（図1参照）内の単方向転送であり，専用化して高速に転送可能である．そこで，ドローイング部へのデータ転送専用の別バスをシステムに用意すれば（図 6，図8参照），ドローイング部の

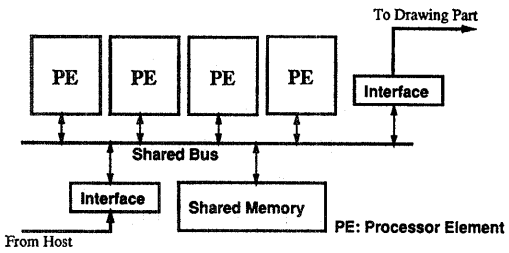


図 5 シングルバスの構成
Fig. 5 Single-bus configuration.

将来の性能向上に対して，ジオメトリ演算部はプロセッサ数の増加で対応できる．

4章や5章でも述べたように，局所性を利用するために各プロセッサがデータキャッシュを持つことを前提にしている．共有メモリ共有バス型マルチプロセッサではデータのコヒーレンシを保つために通常スヌープキャッシュ[14]が用いられる．

8. TSL の管理を支援する機構

4章の最後に述べたようにマルチプロセッサで順次振り分け方式を用いてジオメトリ演算を並列処理すると，PHIGSのデータ構造から派生する制約としてTSLをプロセッサごとに管理する（プロセッサごとに TSL のスナップショットを用意する）必要がある．

最もナイーブな解決法は，共有メモリ上に共通の TSL を設け，各プロセッサがプリミティブの処理開始前に毎回共通 TSL をローカルなメモリエリアにコピーすることでローカルな TSL のスナップショットを確保し，1個のプリミティブの描画中はそのコピーを参照するという方法が考えられる．この方式では，プリミティブの描画のたびに TSL（graPHIGS[7] では約 1K Byte）のコピーが必要となり，コピー自体がオーバヘッドであり，また，共有バスの競合を招き性能が低下する．

結局，この TSL コピー方式や4章の最後で言及した共通 TSL 方式のいずれを用いても，プロセッサの台数増加に伴う性能向上のスケーラビリティを期待することは難しい．従来の順次振り分け方式のシステム[9),10)]はこの問題点を抱えていた．

以上の問題点を解決すべく，トラバーサルを行う主体によって二つの場合に分けて，TSL のメンテナンスの問題を解決する方式とその支援機構を本章において提案する．

8.1 ディスパッチプロセッサ方式と支援機構

構造体トラバーサルとプリミティブのディスパッチを特定の1台のプロセッサ **DP** (Dispatch Processor) が行う場合の方式と機構について最初に述べる．

図 6 のように属性設定の指示を全プロセッサにブロードキャストするバス（**ABUS**：Attribute BUS）を設け，各要素プロセッサに TSL のメンテナンスのためにデュアルポートメモリ3組からなる TTS (Triple Tsl Store) を用意する．ただし，共有バスのデータ転送能力に余裕がある場合は共有バスに ABUS の役割を兼ねさせることもできる．DP は CSS 内のデータのトラバーサルを行い，要素が属性設定の指示であれば，その情報を ABUS に出力し，プリミティブ描画の指示の場合は，処理を行っていない要素プロセッサにそのプリミティブの処理を依頼する（タスクのディスパッチ）．

各 TTS の3組のメモリのうちの1組がマスタ (master) であり，マスタは常に ABUS の内容を反映し最新の TSL の値を保持している．残りの二つのメモリはスレーブで交互にマスタのある時点のスナップショットとして要素プロセッサから参照される．

この機構の動作の概略を述べる．要素プロセッサ PE1 がプリミティブをスレーブメモリの一方である slave1 を利用して処理していると仮定する．slave1 は ABUS から切り離され内容は不変に保たれている（ロックされている）．PE1 がこの処理を継続中の ABUS 上の属性設定の指示は master ともう一方のスレーブ slave2 に反映される（**図 7**）．処理が終了すると PE1 は DP にタスクの処理の終了を通知し，新しいプリミティブの割当を要求する．これと同時に slave1 はロックが解除され，TSL の最新の値をキャッチアップするために，master から slave1 へのコピーが行われる．このコピーの最中も ABUS 上の属性設定の指示

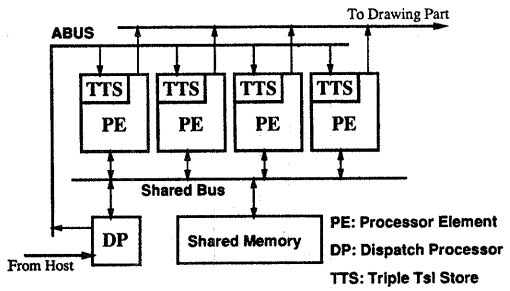


図 6 DP 方式に対応する機構を含む構成
Fig. 6 A system with the mechanism for DP method.

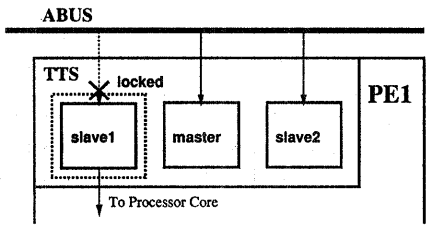


図 7 要素プロセッサ内の TTS の構造
Fig. 7 Structure of TTS in a processor element.

が master や slave1 に反映されるように，デュアルポートメモリが TTS の構成要素に用いられる．slave 1 上で master からの書き込みと ABUS からの書き込みが同じアドレスで競合した場合は，ABUS からの書き込みが優先される．

master から slave1 へのコピーの間は PE 1 に対するプリミティブの新規の割り当てを禁止すれば，slave 2 は必要がない．しかし，新しいプリミティブが前の処理の終了した要素プロセッサ上ですぐに処理可能なように TTS は三重化されている．DP は PE 1 に新しいプリミティブを割り当てる直前に，ABUS を用いて PE 1 に対するスレーブメモリのロック指令を放送する．PE 1 以外の要素プロセッサはこの指令を無視するが，PE 1 は直前に使用されていたスレーブと異なるスレーブメモリ（この場合 slave2）にロックをかける（ABUS からの書き込みを停止する）．この直後に DP が PE 1 に新しいプリミティブの処理を開始させ，PE 1 は slave2 内の属性値を参照しながら処理を行う．PE 1 が新しいプリミティブの処理を行っている間も，master から slave1 へのコピーを継続できる．

TSL のサイズを 1 K Byte とし，マスタからスレーブへのデータコピーの転送レートを 64 bit/クロックとすると，TSL 全体のコピーは 128 クロックで完了する．5 章で述べたように，三次元でシェーディングを伴うプリミティブの処理時間は少なくとも 300 クロック程度かかるので TSL コピーの時間的オーバヘッドはプリミティブの処理の裏処理として完全に隠される．

8.2 シンメトリ方式と支援機構

本節では，専用ハードウェアである TTS を用いずに，汎用アーキテクチャで解決する方法について考える．基本的には特定のディスパッチ用のプロセッサを持たず，各要素プロセッサが独立にトラバーサルを行う方式である．

プリミティブのディスパッチの方式を概説する．共有メモリ上に排他的にアクセスされるカウンタを設け，各要素プロセッサはカウンタの値を読みだし 1 だけインクリメントして更新する（この操作は排他的に行われる）．要素プロセッサは読みだしたカウンタの値と同じ通し番号のプリミティブまでトラバーサルを行い，そのプリミティブの処理を行う（もちろん，TSL は各要素プロセッサごとにトラバーサルの進展に従って管理する）．処理が終わった後，次の処理のプリミティブの番号を得るために再び共有カウンタをアクセスする．この方式で，問題となるのは複数のプロセッサが共有メモリ内の構造体を使ってトラバーサルを行うことに起因するバストラフィックの増加とそれに伴うバス競合である．これを解決する方法として筆者らの考案したスヌープキャッシュ制御機構[15)~17)]を用いることができる．CSS 内のトラバーサルと TSL の管理に必要なメモリ領域に対して，スヌーププロトコルを all-read[16)] に設定しておき，ある要素プロセッサが共有バスを用いてこのタイプのデータを読みだした場合にバスをスヌープしている他のプロセッサのキャッシュもそのデータをできる限りキャッシュ内に取り込むようにする（図 8 参照，右端のプロセッサがバスアクセスを行っている）．これにより，他の要素プロセッサがトラバーサルのために同じデータをその後に読もうとした場合，そのデータはキャッシュされていることが期待され，共有バスへのアクセスの必要もなく，また高速にデータを参照できる．ただし，あるプリミティブの処理を終えた後，新しいカウンタの値に対応するプリミティブを獲得するまでのトラバーサルによる空読み（言い換えると，他のプロセッサに割り当てられたプリミティブを読み飛ばすためのポインタチェイス）は時間的オーバヘッドになる．

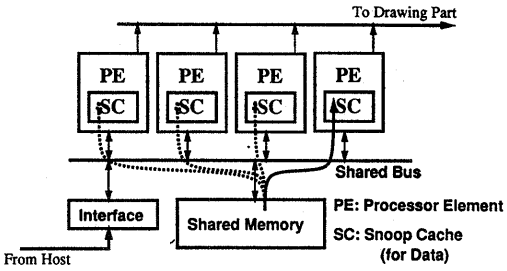


図 8 all-read プロトコルの動作
Fig. 8 Behaviors of 'all-read' protocol.

9. おわりに

PHIGS を採用した高性能三次元グラフィックスシ

ステムの構築を可能にするために，高性能化のボトルネックになるジオメトリ演算に対する並列処理方式について検討を行った．まず，プロセッサの使用効率などの面から均質型マルチプロセッサがパイプライン構造の並列処理より優れていることを定性的に示した．次に，定量的な検討を加えるためにプリミティブ単位のジオメトリ演算部内での処理量とデータ転送量を見積もり，さらにシステム全体の目標性能とジオメトリ演算部に要求される性能の関係を評価した．その結果，基本的にジオメトリ演算がコンピュテーションインテンシブであるため内部のデータ転送路に構造の簡単なバスを使用できることを示し，プログラミングモデルの単純さなどから共有メモリ型の優越性を述べた．そして，共有メモリ共有バス型マルチプロセッサをジオメトリ演算部に採用する場合には，PHIGS 特有のデータ構造から派生するプロセッサごとの TSL 管理が問題となることを指摘し，この問題を解決する方式とハードウェア支援機構を提案した．プリミティブのプロセッサへの割り付けのオーバヘッドが性能へ及ぼす影響，トラバーサルの順次性からくる性能の上限値，バス競合の性能への影響等については，実行駆動型のシミュレータを作成して定量的な評価を行っている．この結果についても別途公表する予定である．

謝辞 初期の議論に参加して貴重な助言をいただいた東京工業大学計算機センターの太田昌孝氏，現在の実装および半導体テクノロジに関してアドバイスをいただいた間嶋勇氏をはじめとする IBM 野洲研究所の方々に深く感謝します．また，PHIGS に関してご指導いただいた IBM 東京基礎研究所の清水和哉氏と浦野直樹氏に謝意を表します．

参 考 文 献

1) Poulton, J. et al.: PIXEL-PLANES: Building a VLSI-Based Graphics System, *Proc. of 1985 Chapel Hill Conf. on VLSI*, pp. 35-60 (1985).
2) Uttley, V.: A New Generation of Graphics Performance, 情報処理学会マイクロコンピュータとワークステーション研究会報告, 63-3 (October 1990).
3) Gharachorloo, N. et al.: A Million Transistor Systolic Array Graphics Engine, *Proc. of 1988 Int. Conf. on Systolic Arrays*, pp. 193-202 (1988).
4) Nishizawa, E. et al.: A Hidden Surface Processor for 3-Dimension Graphics, *Proc. of 1988 IEEE Int. Solid-State Circuits Conference*, pp. 166-167 (February 1988).
5) Haeberli, P. and Akeley, K.: The Accumulation Buffer: Hardware Support for High-Quality Rendering, *ACM Computer Graphics*, Vol. 24, No. 4, pp. 309-318 (1990).
6) Information Processing System—Programmer's Hierarchical Interactive Graphics System (PHIGS) Part 1—Functional Description, ISO, DP 9592/198n (E) (October 1986).
7) The graPHIGS Programming Interface Understanding Concepts Version 2, Release 1.0, IBM Corp. (1989).
8) Akeley, K. and Jermoluk, T.: High-Performance Polygon Rendering, *ACM Computer Graphics*, Vol. 22, No. 4, pp. 239-246 (1988).
9) Kirk, D. and Voorhies, D.: The Rendering Architecture of the DN10000VS, *ACM Computer Graphics*, Vol. 24, No. 4, pp. 299-308 (1990).
10) Horning, R. et al.: System Design for a Low Cost PA-RISC Desktop Workstation, *Proc. of COMPCON 91 SPRING, IEEE*, pp. 208-213 (February 1991).
11) 杉沼浩司：コンセプトを競うグラフィックス・ハードウェア，PIXEL, No. 98, 図形処理情報センター，pp. 128-132 (1990).
12) Fisher, J. A.: Very Long Instruction Word Architectures and the ELI-512, *Proc. of 10th Int. Symp. on Computer Architecture*, pp. 140-150 (1983).
13) Cohen, E. et al.: Discrete B-Splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics, *Computer Graphics and Image Processing*, Vol. 14, pp. 87-111 (1980).
14) Papamarcos, M. and Patel, J.: A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories, *Proc. of 11th Int. Symp. on Computer Architecture*, pp. 348-354 (1984).
15) 松本 尚：細粒度並列実行支援機構，情報処理学会計算機アーキテクチャ研究会報告，77-12, pp. 91-98 (July 1989).
16) 松本 尚：細粒度並列実行支援マルチプロセッサの検討，情報処理学会論文誌，Vol. 31, No. 12, pp. 1840-1851 (1990).
17) Matsumoto, T. et al.: MISC: A Mechanism for Integrated Synchronization and Communication Using Snoop Caches, *Proc. of the 1991 Int. Conf. on Parallel Processing*, Vol. 1, pp. 161-170 (August 1991).

付　　録

Polygon3, Triangle Strip, NURBS 曲面の処理時間（クロック数）の内訳をそれぞれ**表 4, 5, 6** に示す．

表 4 Polygon3 の処理時間の内訳
Table 4 Items of processing time for Polygon3.

処　理　内　容	時間 (clock)	
座標変換 (MC→WC)	39	
照度計算	359	
法線計算		53
視線ベクタの正規化		43
主計算 (3 Positional)		251
その他		12
座標変換 (WC→NPC)	66	
クリッピングチェック	36	
座標変換 (NPC→DC)	39	
後処理	30	

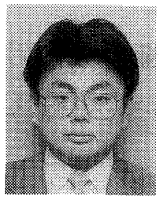
表 5 Triangle Strip の処理時間の内訳
Table 5 Items of processing time for Triangle Strip.

処　理　内　容	時間 (clock)
座標変換 (MC→WC)	20
視線ベクタの正規化	36
法線ベクタの計算	56
照度計算	
1 Positional	138
3 Positional	(268)
3 Positional+3 Directional	(434)
座標変換 (WC→NPC)	42
クリッピングチェック	12
座標変換 (NPC→DC)	15
後処理	20

表 6 NURBS 曲面の処理時間の内訳
Table 6 Items of processing time for NURBS.

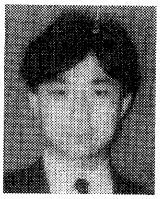
処　理　内　容	時間 (clock)	
テセレーション	181	
頂点座標計算		42
法線計算（正規化）		139
視線ベクタの正規化	43	
照度計算		
1 Positional	138	
3 Positional	(268)	
3 Positional+3 Directional	(434)	
座標変換 (WC→NPC)	42	
クリッピングチェック	12	
座標変換 (NPC→DC)	15	
後処理	40	

(平成 4 年 8 月 24 日受付)
(平成 5 年 10 月 14 日採録)



松本　尚（正会員）

1962 年生．1985 年東京大学工学部計数工学科卒業．1987 年大阪市立大学大学院理学研究科物理学専攻修士課程修了．日本アイ・ビー・エム(株)東京基礎研究所研究員を経て，1991 年 11 月より東京大学理学部情報科学科助手．並列計算機アーキテクチャ，オペレーティングシステム，最適化コンパイラに関する研究に従事．他にニューラルネットワーク，学習，力の超統一理論等に興味を持つ．1990 年本学会学術奨励賞受賞．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員．



川瀬　桂

1961 年生．1985 年早稲田大学理工学部機械工学科卒業．1987 年同大学院前期課程修了．同年日本アイ・ビー・エム(株)に入社．現在同社東京基礎研究所に勤務．グラフィックスの並列処理の研究に従事．

森山　孝男（正会員）

1962 年生．1985 年東京工業大学工学部情報工学科卒業．1987 年同大学院修士課程修了．同年日本アイ・ビー・エム(株)に入社．現在同社東京基礎研究所に勤務．並列マシンのオペレーティングシステム，グラフィックスの並列処理の研究に従事．