グラフィクスとCAD 16－1
（1985. 3. 12）

多面体のリアルタイム表示装置

polyhedron display in real-time

大野哲朗　杉原厚吉　杉江昇
Tetsuo OHNO　Kokichi SUGIHARA　Noboru SUGIE

名古屋大学工学部
(Faculty of Engineering, Nagoya University)

This paper describes a hardware architecture and a computer algorithm of a multi-proccessor system for real-time display of 3-D polyhedrons under a parallel light source.
Since this system is designed in such a way that polyhedrons are proccessed in parallel for 2-level hidden-surface removal, the system architecture is natural and very simple. Each proccessor element proccesses only one convex polyhedron and generates descriptions of visible faces in the format segmented lines, which are written in a spetial type of frame-buffer.

1. はじめに

リアルタイムの3次元図形表示は、最初はワイヤーフレーム表現に対して実現された。(1) その後、ラスターグラフィックス処理のハード化の研究は、多数行なわれたが(2) リアルタイムな表示を目標としたものには、画面を分割した並行処理(3)などがある。リアルタイムのラスターグラフィックスを必要とする応用では画質はともかく、とにかくリアルタイムで表示できるものが欲しいという要求もあるだろう。たとえば、表示物体は多面体で、照明は平行光線であるというような制限にも、ロボットマニュピレータのシミュレーションなどには、十分役立つであろう。

本研究では、こうした制限された3次元物体をリアルタイムで表示するシステムを試作した。これは、対象物体の特質をうまく利用すると同時に、システムの構成を単純なものにし、拡張性のある構成を目指したものである。すなわち、対象を凸多面体に分割し、各凸多面体ごとに並列処理できる部分を専用のプロセッサで行ない、最後に簡単な処理で全体の処理をまとめるというものである。

2. システムの基本概念

3次元図形表示のプロセスは、座標変換、クリッピング、隠面消去、輝度計算フレームバッファの書き換えの順に行なわれる。ここでは、対象物体を平行光線下の多面体に限定しているので、座標変換、クリッピング、輝度計算は、面単位で処理できる。しかし、隠面消去とフレームバッファの書き換えは、画素単位の処理なので、扱うべきデータ量が多いため処理に時間を要する。そこで、この部分を並列処理化させて、処理速度を上げようというのが通常の手段である。

本研究でも、座標変換、クリッピングなどは、ホストコンピュータで処理してそれ以降の処理、隠面消去と輝度計算そして、フレームバッファの書き換えを専用の並列処理化したハードウェアにまかせることにした。

3．隠面消去アルゴリズムと並列構造

隠面消去は、次の２段落で行う。

1．凸多面体単位の隠面消去（面単位の処理）
　１個の凸多面体に着目すると、全ての面は、完全に見えるか、完全に見えないかのどちらかである。よって、１つの凸多面体のみでは、裏の面を除去するだけでよい。

2．凸多面体間の隠面消去（画素単位の処理）
　凸多面体の表の面を走査変換して、輝度とZ値をもった画素に展開する。そして、各凸多面体間で画素のZ値を比較し，最も小さいZ値をもつ画素、すなわち最も手前にある画素を選択して、隠面消去を実現する。

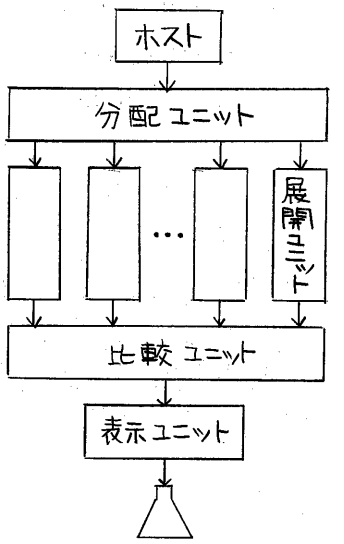

図1．システムブロック図

このアルゴリズムは、並列処理で実行される。つまり、凸多面体単位の隠面消去と表の面の走査変換は、各凸多面体ごとに割り当てられたプロセッサで行なわれ、最後に、各凸多面体の画素がCRTのビームスキャンに同期して読み出され比較される。すなわち、CRTのリフレッシュと同時に隠面消去が行なわれるわけである。

4．システム概要

３次元物体表示プロセスのうち、隠面消去，輝度計算、CRTへの表示を凸多面体単位で、並列処理する。図１にシステムブロック図を示す。ホストコンピューター内の図形データベースには、３次元物体が凸多面体単位で入っている。アプリケーションプログラムにより操作された３次元物体は、凸多面体ごとに座標変換、クリッピングが行なわれ、３次元物体表示ユニットに送られる。

３次元物体表示ユニット内では、分散ユニットにより、各凸多面体に１つずつ割り当てられた展開ユニットにデータが送られ、そこでは、凸多面体単位での隠面消去、すなわち不可視面の除去と面の輝度計算が行なわれる。そして、輝度とZ値をもつ画素に走査変換し、それを、CRTのリフレッシュに従って比較ユニットに送る。比較ユニットでは、全ての展開ユニットから送られる画素のZ値を比較し、最も手前にある（すなわち最もZ値の小さい）画素の輝度を表示ユニットに送る。そして、表示ユニットは、輝度をD/A変換し、CRT上に表示する。

各ユニットは、それぞれ独立に動作し、すべてパイプライン化されて，見かけ上の処理速度を上げている。

5．表示物体のデータ構造

表示物体のデータ構造は、表１のとおりである。すなわち、対象物は、凸多面体から成り、各凸多面体は、面・頂点・稜線より成っている。一般には、稜線デ

ータは、不必要であるが、これは、後で述べる展開ユニットで利用される。凸多面体を構成する面・頂点・稜線データの中でフレームごとに変化するのは、頂点のみであり、面・稜線データは、不変である。したがって、面と稜線データは、初めて、凸多面体が、展開ユニットに割り当てられる時だけ送り、それ以降は、変化する頂点データと視点座標のみを送りさえすればよい。

多面体 := 凸多面体へのポインタリスト
凸多面体 := { 面へのポインタリスト / 稜線群へのポインタリスト / 頂点座標リスト }
面 := { 色（R-G-B） / 頂点ポインタリスト（表から見て反時計回りに並んでいる） }
稜線群 := 1つの頂点から出ている稜線のリスト
稜線 := { 終点頂点へのポインタ / 右側面へのポインタ / 左側面へのポインタ }

［表1. 表示物体のデータ構造］

6. 展開ユニット

各々の展開ユニットが、1つの凸多面体を処理する並列構造をなしている。その役割は、凸多面体単位での隠面消去と面の輝度計算、そして、輝度とZ値をそつ画素データに凸多面体を走査変換し、それをCRTのビームスキャンに同期して、次の比較ユニットに送り出すことである。ユニット内部のブロック図を図2に示す。

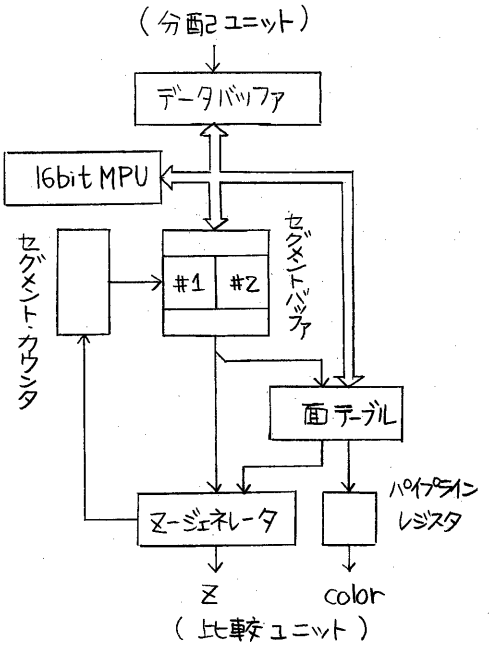

図2 展開ユニット ブロック図

(1) 凸多面体データの入力

データバッファは、2Kワードの容量をもつ共有メモリである。内部バスを通して、16bit MPUに、そして、分配ユニットを介して、ホストにマップされている。特別なアービターは、備えておらず、ホストがアクセスする時は、展開ユニットのMPUを停止させることで、アクセス競合をさけている。このデータバッファには、初期化時（初めての凸多面体が割り当てられた時）に、頂点とともに、面・稜線データがセットされる。そして、ユニットの初期化ルーチンでこれを、利用しやすい形に変換し、以後の処理（主に凸多面体の走査変換）に利用する。

(2) 凸多面体単位の隠面消去

凸多面体の面は、完全に見えるか、完全に見えないかのどちらかであるから、面の向きを調べて裏向きならば、データバッファ内の面データを削除し、以後の処理から除く。

面の向きの判定は、面を構成する隣り合った3頂点の座標から法線ベクトルのZ成分を算出して行なう。法線ベクトル$\vec{N}$は、

$$\vec{N} = (\vec{P_3} - \vec{P_2}) \times (\vec{P_1} - \vec{P_2})$$

よって、

$$Nz = (x_1 - x_2)(y_3 - y_2) - (x_3 - x_2)(y_1 - y_2) \quad \cdots (1)$$

となる。面データの頂点ポインタリストは、面の表から見て、反時計回りに並んでいるので、この頂点ポインタリストの第1，2，3頂点を使えば、(1)式の正負より、面の向きを判定できる。

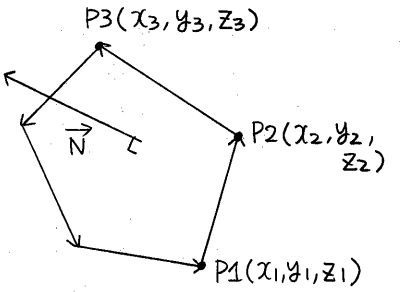

図3. 面の向き判定

(3) 面の輝度計算

照明が平行光線であるから、各面内の画素は、位置に関係なく同じ輝度をもつ。よって、表向きの面のみの輝度を面ごとに一度計算すればよい。輝度は、光線ベクトルと面の法線ベクトルの内積に比例するので、その値により、面データ内の固有輝度R-G-Bを変える。

(4) 面のセグメント表現

ここでは、画素データの語長が、RGB各8bit，Z16bitの計40bitとあるので、通常の画素単位のフレームバッファでは、大容量のメモリが必要になってしまう。そこで、画素データをセグメント化することでデータの大幅な圧縮をなした。セグメントというのは、図4のように、走査線を含みXZ平面に平行な面とポリゴンとの交線である。このセグメントは、不定長であるので、画素単位によるフレームバッファのようなランダムな書き込みはできない。そこで、ソフトウェアにより、凸多面体をスクリーン座標系の左上から右下へと、セグメント単位で走査変換しシーケンシャルにセグメントバッファに書き込む。そして、次に、Zジェネレータによりセグメントを CRTのリフレッシュに同期して、輝度とZ値をもつ画素に改めて、展開し送り出す。

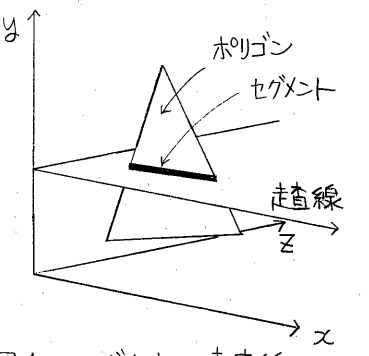

図4 セグメントと走査線

セグメントバッファは、言わゆる「ダブルバッファ」であり、MPUが一方のバッファに走査変換したセグメントを書いている間に、もう一方のバッファのセグメントをZジェネレータが画素に展開していく。そして、それが、交互に切りかわるわけである。

セグメントデータは、できるだけ圧縮し、また、Zジェネレータによる画素への展開にも都合のよいものにしなければならない。セグメントのデータ構造を図5に示す。セグメント自身は、1ワード16bitで2ワード構成である。第1ワードには、セグメントの始点のZ座標（Z0；15bit）があり、第2ワードには、セグメントの正射影されたX軸方向の長さ（DX；9bit）と、このセグメント

が属する面の番号（ＰＮＵＭ；６bit）よりなっている。各ワードとも１bitのＳＹＮＣというタグフィールドを持っている。これは、走査線の開始を示す。つまり、それぞれの走査線の最初のセグメントはＳＹＮＣ＝１となり、それ以降のセグメントではＳＹＮＣ＝０となっている。始点座標が、Ｚのみでいいのは、各走査線には、少なくとも１つのセグメントがあり、ＳＹＮＣフィールドにより、現在どの走査線上にあるかが分っているからである。また、凸多面体の左側のブランク部分もセグメントで表わすことにより、Ｘ座標を必要なくなる。これは、セグメントは、全て、逐に走査変換されており、さらにＤＸすなわち走査線上の長さも分っているから、Ｘ座標は、インクリメンタルな計算で算出できるわけである。さらにセグメントの輝度とＺ方向の傾き $\Delta Z/\Delta X$ は同じ面に属していれば、同じなので、各セグメント自身が持つのは、不経済である。そこでＰＮＵＭという面番号のみを持ち、これで輝度と傾きの入った面テーブルを参照する。

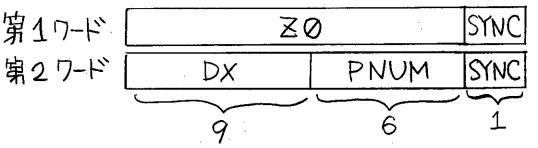

図５　セグメントのデータ構造

(5)　凸多面体の走査変換

凸多面体の可視面をセグメント単位に走査変換する処理は、ＭＰＵにより、ソフトウェアで実現されている。そして、この処理が、展開ユニットにおける処理時間のかなりを占めている。

セグメント単位の走査変換というのは、つまり、セグメントをＣＲＴの表示画面といえば、左上から順に右下まで走査方向にソートすることである。これは、稜線をＹ方向，Ｘ方向にソートしていくことにより行なう。そして、ここで、ホストでは冗長であった稜線データが役立つ。すなわち、稜線データにより、各頂点からどの頂点に稜線が出ているかをただちに知ることができる。

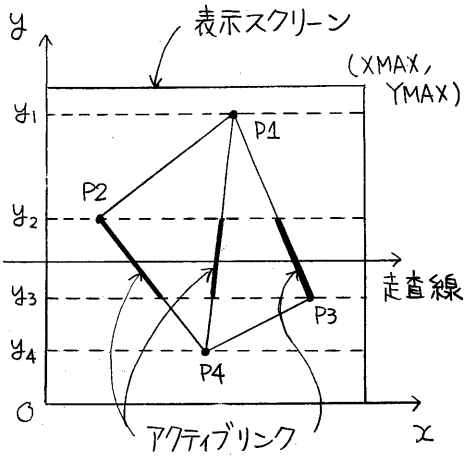

図6.　スキャンラインコヒーレンス
とアクティブリンク

また、処理効率を向上させるために、スキャンラインコヒーレンスを利用する。たとえば、図６で，走査線が、頂点Ｐ２とＰ３の間にある間は、稜線の関係は一定である。つまり、各走査線のセグメント構成は、まったく変らず、ただセグメントの長さがインクリメンタルに変化していくだけである。これをスキャンラインコヒーレンスと呼ぶ。またここで、走査線と交れる稜線をアクティブリンクと呼ぶことにする。よって、Ｐ２からＰ３の間では、一度アクティブリンクが得られれば、インクリメンタルな計算のみで、ソートされたセグメントを生成できる。

凸多面体の走査変換は、次のように行なわれる。

１．視点から見えない頂点を除去し、可視面のＸ軸に沿ったＺ方向の傾き $\Delta Z/\Delta X$

を算出する。

2. 全頂点をyでソートし，y-sortリストをつくる。
3. 最初の頂点が、画面の一番上になければ、ブランクを表示するセグメントを生成する。
4. 次の頂点までのアクティブリンクを稜線データから取り出し、x方向にソートしたアクティブリンクリストをつくる。このとき、新しくアクティブになった稜線の傾き $\Delta x/\Delta y$，$\Delta z/\Delta y$ を計算して、稜線データに入れる。また、アクティブでなくなった稜線は、稜線データから除去する。
5. アクティブリンクリストに従って、セグメントを生成する。各セグメントの始点と長さは、稜線データ内の傾き $\Delta x/\Delta y$，$\Delta z/\Delta y$ によりインクリメンタルに計算でき、属する面も稜線データ内の右側面ポインタにより得ることができる。
6. 4,5の処理をy-sortリスト中のすべての頂点について行なう。
7. 最後の頂点が、画面の一番下になければ、ブランクを表示するセグメントを生成する。

(6) 画素展開とZジェネレータ

走査変換されたセグメントは、Zジェネレータで輝度とZ値をもつ画素に展開される。Zジェネレータ周辺のブロック図を図7に示す。画素展開は、まず、セグメントデータの第1ワードのZ0がZレジスタにロードされ，次に第2ワードのDXがXカウンタに、PNUMの値により、面テーブルの $\Delta z/\Delta x$ がDEFFレジスタにロードされることではじまる。そして、X-カウンタが－1されるごとに

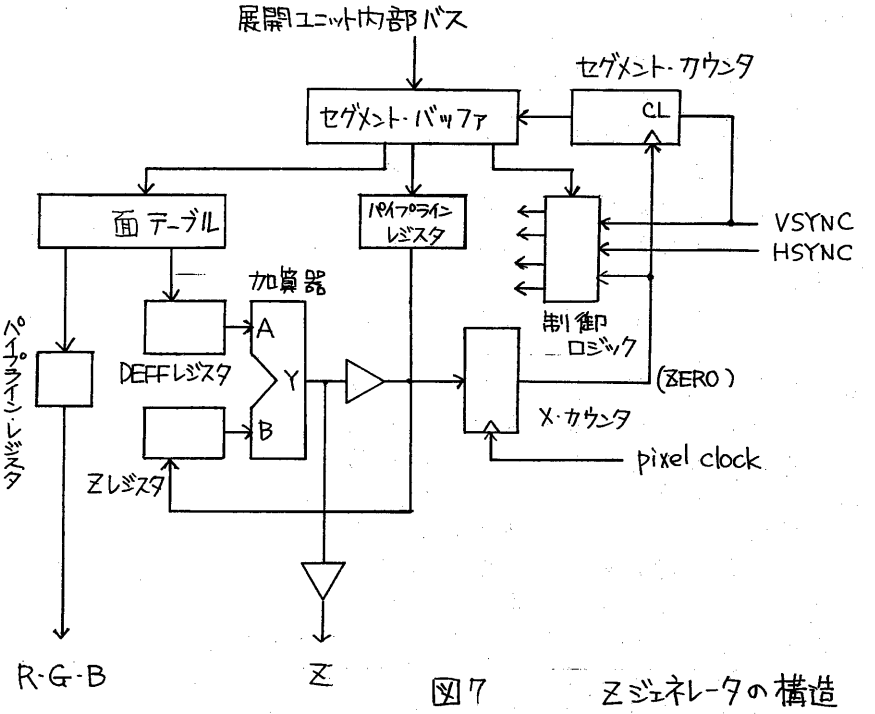

図7　Zジェネレータの構造

DEFFレジスタとZレジスタが、加算され、結果がZレジスタに再びロードされる。これがXカウンタが0になるまで続き、その間，1画素ごとに輝度とZ値が出力されるわけである。

7. 比較ユニット

画素データは、RGB各8bitとZ16bitで展開ユニットからCRTのリフレッシュに同期して送られてくる。凸多面体間の隠面消去は、この画素のZ値を比較して、最も小さいZ値をもつ画素のみをCRT上に表示することによって実現される。

Zの比較は、2つずつ、比較セルユニットで行なわれ、それをTree状に並べることにより、全ての展開ユニットの比較を実現している。各比較セルユニットは、1画素単位でパイプライン化されており、見かけ上、全ての比較が1画素時間で行なわれることになる。（図8参照）

図9に比較セルユニットの構造を示す。画素データ1，2のZ値は、コンパレータで比較され，Zの小さい画素がマルチプレクサを通って、パイプラインレジスタにラッチされ、次段の比較セルユニットに送られる。

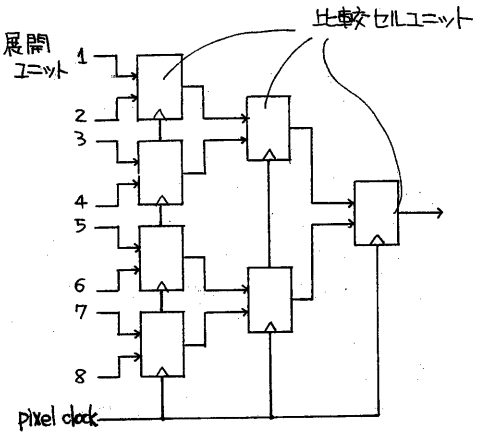

図8　パイプライン化されたTree状比較ユニット

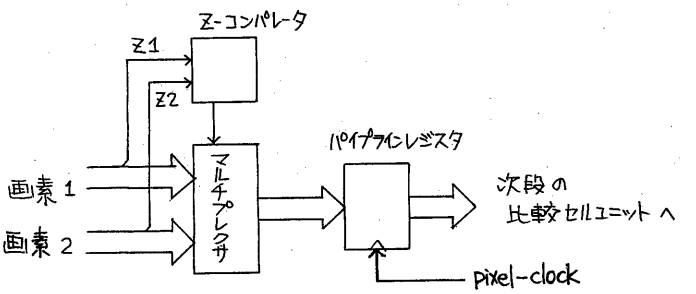

図9　比較セルユニットの構造

8. システムの評価

展開，比較ユニットを1つずつ試作し、その能力の評価を行った。本来ならば、展開ユニットを2個以上作り、凸多面体間の隠面消去を行なわすべきではあるが、ここでは、比較ユニットに特定のZ値を与えることにより、ある奥行きにあるXY平面に平行な面を作り出し、それと凸多面体との隠面消去により評価することにした。ホストには、16bitパーソナルコンピュータPC9801Fを用いた。凸多面体が1つであるから、なんとかリアルタイム処理が行える。

その結果であるが、10面程度の凸多面体ならば、1/20秒で表示できた。15面以上では、ホストの処理が間に合わなくなる。

今回、試作したハードウェアは、展開，比較ユニットが、15cm×23cmのユニバーサル基板2枚にラッピングによる配線で実装されている。そして，LSI

（ＣＰＵやメモリなど）とＬＳ-ＴＴＬ（一部にＳ，ＦＡＳＴの高速ＴＴＬ）など合わせて約１００個のロジックで構成されている。

9. まとめ

本研究では、対象物体を多面体，照明を平行光線に限定した上で，多面体を凸多面体単位に分割し、並列処理させることで，３次元物体のリアルタイム表示を行なうということを提案し，そのための ハードウェアシステムを試作した。

この凸多面体分割による並列処理は、非常に単純なシステム構成を可能にする。それは、凸多面体分割が、表示物体に対して非常に自然だからである。そのため、システムは、凸多面体を処理するプロセッサを単に集めただけで済ますことができる。しかし、この単純な構成は、逆に言えば、並列システム全体の処理能力を最大限に使うことができない。また、同時に表示できる凸多面体の数は、並列プロセッサの数に制限される。しかし、１つのプロセッサに互い重ならない複数の凸多面体を処理させることで、表示能力をある程度増すことはできるであろう。

リアルタイム表示では、フレームコヒーレンスをうまく利用すれば、大幅な処理の軽減が期待できる。このシステムでは、１つのプロセッサが、１つの凸多面体をずっと処理するので，このフレームコヒーレンスが利用しやすいわけである。今のところ、面や稜線データをプロセッサがもつことにより、凸多面体の走査変換に利用しているにすぎないが、この特性の利用について、さらに考えて行きたい。

最後に、このようなシステムが、有効なのは、ロボットアームの動作シミュレーションなどのように、数十個程度のプロセッサによるごく簡単な物体のリアルタイム表示であろう。その程度ならば、かなり低コストでシステムが実現できるはずである。

〔参考文献〕

(1) Newmann, W.M., and Sproull, R.F.; "Principles of Interactive Computer Graphics, 2nd Edtion", McGraw-Hill, 1979.

(2) Sutherland,I.E, R.F.Sproull, and R.A.Schumacker; "A Characterization of Ten Hidden-Surface Algorithms," ACM Computing Surveys (Mar. 1974)

(3) 今井ほか、"三次元色彩図形表示プロセッサのハードウェア構成" 情報計算機アーキテクチャ研資料，50-4，１９８３,９