修 士 論 文

仮想現実環境における衝突検出ハードウェアに関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

松本 健太郎

2005 年 3 月

修 士 論 文

仮想現実環境における衝突検出ハードウェアに関する研究

指導教官　井口 寧 助教授

審査委員主査　井口 寧 助教授
審査委員　松沢 照男 教授
審査委員　堀口 進 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

310106 松本 健太郎

提出年月: 2005 年 2 月

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の背景と目的

近年、コンピュータの高速化や普及に伴って、仮想現実 (virtual reality) が実用化され産業分野で使われるようになってきた。コンピュータゲーム、医療シミュレーション、ロボットの遠隔操作など、応用範囲は様々である。

現在の仮想現実は視覚的な現実性に重点をおいているものが多く、オブジェクトの衝突 (干渉) についての現実感はまだ多くは無い。従来から衝突検出の多くはオブジェクトを単純な球やボックスで囲んだり近似したもが多く、複雑なオブジェクト同士の衝突や多数のオブジェクト環境における衝突検出はあまり行われてこなかった。

他方で最近、産業界において製品の企画設計段階に仮想現実技術を取入れることで、開発にかかる時間や金銭コストの削減が望まれている。没入型 6 面ディスプレイシステム COSMOS[3] には自動車や飛行機などの 3D モデルを実物大で表示できる利点があるが、企業からは” モデル同士の干渉を検出して欲しい” という要望が常に言われてきた。この様な要望に答えようと、仮想現実空間で仮想オブジェクトを触ったり移動したりできるシステムの研究がされている。この研究では、物体同士の干渉に現実性を持たせるために仮想オブジェクトの干渉検出にポリゴンをもちいている。また、ポリゴンの量が多いため衝突 (干渉) 計算量を減らす目的で境界ボリュームを用いることでリアルタイム性を実現している。この研究に限らず現在行われている衝突検出の研究の大半が境界ボリュームを使った計算量削減がである。衝突検出の計算量を削減する主な理由はオブジェクトを構成するポリゴン数が年々増加する傾向にあることと、衝突検出の計算量がポリゴン数 n に対して $O(n^2)$ だからである。従って計算量削減は重要であるが、計算量の削減に片寄ったシステムでは、オブジェクトの変形や破壊などに弱いという研究結果がある [4]。これは事前計算で計算負荷の高い境界ボリュームを階層的に作り上げているため、変形すると再度、高負荷なボリュームを構築し直さなければならないからである。

仮想現実環境において、オブジェクトの変形は実現すべき機能なので、これに対応できる衝突検出システムが望まれる。そこで、計算量の削減は少ないが境界ボリュームの構築に手間がかからないものとして、境界ボリュームに比較的単純な AABB(Axis-aligned bounding box)[1] を使ったのもや、GPU(graphics processing unit)[5] の可視性チェック機能を使って計算量の削減を高速に行う試みが行われている。これらの利点としては階層型と比較して高速であることだが、従来より三角ポリゴンの削減数が減り、交差検出処理時間が多くかかるという課題がある。この問題解決に三角ポリゴンの交差検出処理の高速化が望まれている。

そこで、本研究では三角ポリゴンの交差検出処理を高速化する目的で、専用ハードウェアの設計を行い、更に境界ボリュームを考慮したハードウェア構成を明らかにする。

1.2　本論文の構成

本論文は 6 章で構成される。第 2 章では、仮想現実環境や、そこで行われる衝突検出について従来研究から幅広く説明する。第 3 章では、三角ポリゴン同士の交差検出を行う専用のハードウェアの要求仕様を明

らかにし、そのハードウェア構成について詳しく述べる。

第４章では、衝突検出処理で一般的に使われている境界ボリュームを使った環境を想定した場合に、効率良く三角ポリゴンの交差検出処理が行えるハードウェアについて記述する。また、第３章で示した三角ポリゴン交差検出ハードウェアの並列化や PC/AT 互換機の PCI デバイス上の FPGA を想定してデータ転送にかかる時間コストなどを 考慮したシステムの構成を示す。第５章では、第３章と第４章で示したハードウェアの FPGA への実装結果と、その性能についてシミュレーション実験により他の研究との評価を行う。第６章では本研究のまとめと今後の課題について述べる。

第2章　仮想現実環境における衝突検出

2.1　はじめに

本章では、仮想現実環境の定義を行い、仮想現実環境において重要な衝突検出処理について従来研究を引用してその重要性を説明する。また、衝突検出処理においてボトルネックになる三角ポリゴン交差検出処理について、従来のソフトウェア処理に用いられているアルゴリズムを比較する。そして、後の 3 章以降のハードウェア設計に用いる三角ポリゴン交差検出アルゴリズムの流れを説明する。

2.2　仮想現実環境

仮想現実 (バーチャルリアリティ) とはコンピュータ・グラフィックスや音響効果などを組み合わせて、人工的に現実感を作り出しすことで、人間の五感に働き掛けて、より現実に似た体験をすることができる技術である。単に「人工的な現実感」といった場合には、例えば小説や映画といったメディア表現も含まれるが、仮想現実の構成要件としては以下の要素が必要とされる。[6]

- 体験可能な仮想空間 (環境) の構築
- 五感のうちいくつかに働きかけて得られる没入感
- 対象者の位置や動作に対する感覚へのフィードバック
- 対象者が世界に働きかけることができる対話性

この基準に照らせば、例えば小説には視聴覚による没入感に欠け、映画には対話性が欠けるため、仮想現実とはみなされない。

バーチャルリアリティ(Virtual Reality) という用語は、1987 年に NASA(米航空宇宙局) が VPL Research 社に発注して開発した「VIEW」(仮想環境ワークステーション) というシステムの開発プロジェクトに際して使い始めた語で、語感の先進的な響きとシンプルさが受け入れられて、コンピュータシステムで現実感を作り出す技術の総称として定着した。

仮想現実システムはコンピュータと入出力機器の組み合わせによって構築される。頭に装着して視界をすっぽりと覆うヘッドマウントディスプレイ（HMD:Head Mount Display) や、手の動きを入力としたり擬似的に触覚を与える手袋上のデータグローブなど、様々な機器が考案され、いくつかは実用化されている。

仮想現実の応用範囲はアミューズメントから産業、医療など様々なものが考えられる。例えば産業分野では多くの企業から 3D モデルの試作段階の製品の” 実物大表示” だけでなく、” モデルに触った感触が欲しい”、“モデル同士の干渉を検出して欲しい” という要望は常に言われてきた。[3] これにより従来試作を製作するまで分からなかった問題点をデザイン、設計段階で解消するだけでなく、試作開発にかかる時間、費用のコスト削減をすることが求められている。

浅野ら [3] は、没入型 6 面ディスプレイシステム (COSMOS) の中で、モデル同士の干渉検出やユーザが直感的にモデルに触れて操作できるような衝突検出システムの開発を行った。衝突検出の方法としてはモデルの形状を外接直方体で近似した方式とモデルを構成するポリゴンデータを利用する方式の 2 つを開発した。

2.3 仮想現実環境における衝突検出

仮想現実環境での衝突検出について述べる。

仮想現実はコンピュータ上で作られる環境であるので、コンピュータ特有の表現方法がある。まず、仮想空間上のオブジェクトはどのようにコンピュータ上で表現されるのか説明する。

そして、コンピュータ上で表現されるオブジェクト同士の衝突とはどの様なものなのか、そして衝突を検出する方法にはどの様な方法があるのか論ずる。

2.3.1 オブジェクトの表現

前説でも述べたが仮想現実ではできる限り現実世界に近いオブジェクトをコンピュータの性能の制約 (処理速度やメモリの容量) 内で表現をする。仮想現実でのオブジェクト表現は主に以下の 2 種類に分けられる。

- NURBS やスプライン・パッチなどの曲面や曲線を組み合わせる方法
- 三角形の平面 (三角ポリゴン) を組み合わせる方法

IBM 社の CATIA[7] などの 3 次元 CAD では滑らかで正確な形状が得られる NURBS が多く使われているが、3D のリアルタイム動画表示を行うゲーム業界や仮想現実技術関係ではポリゴンによる表現が主流である。

三角ポリゴンによる表現

オブジェクトは三角ポリゴンのメッシュによって構成される。その内部表現は頂点座標配列とトポロジ配列から成る。あるオブジェクトの構成要素の 1 つである三角ポリゴン T_1 を例 (図 2.1) に示す。T_1 はトポロジ配列の先頭にあり T_1 を構成する 3 つの 3 次元頂点座標配列のアドレス値 (V_2, V_1, V_6) をもつ。そのアドレス値を元に V_2 の頂点座標 (x_2, y_2, z_2) を取り出し、同様に V_1 と V_6 についても取り出す。この様に 3 回アドレスを指定して 1 つの三角ポリゴンの頂点座標を取出すことができる。通常、1 つの頂点座標は複数の三角ポリゴンと共有することからこの方法をとる。

2.3.2 衝突検出

現実の物理空間では、異なるオブジェクトが同時に同じ空間を共有することはできないので、衝突したら互いにめり込まないような相互作用が起こる。仮想現実環境においても、仮想オブジェクト間に起こる相互作用を再現する必要がある。しかし、仮想現実環境は多数のオブジェクトやポリゴンで構成されており、更にリアルタイム性も求められる為に、全てのオブジェクトに対してこの相互作用を計算することは不可能である。そこで、コンピュータ上の仮想環境では、まずオブジェクト同士の衝突を検出処理を行って、衝突しているオブジェクト・ペアだけに対して相互作用を計算することが行われている。

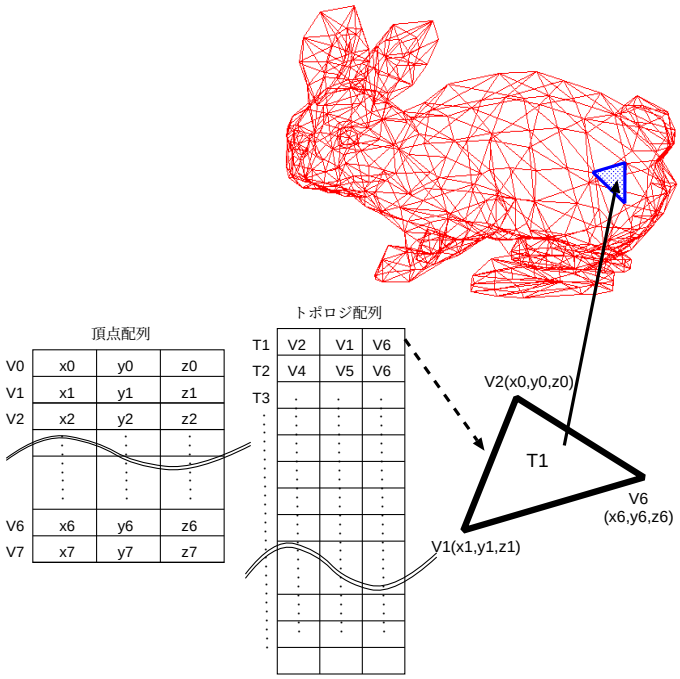


図 2.1: ポリゴンオブジェクトのデータ構造

また、衝突検出は基本的に $O(n^2)$ であるので、仮想環境内のオブジェクト数が特に多い場合は、加速度的に衝突検出回数が増加する。この場合、仮想空間を分割して衝突の可能性をオブジェクトを減らすことで計算量を減らすことがされており、代表的なアルゴリズムとして Octree がある。Octree は３次元空間を８つに分割して、オブジェクトを内包する空間のみ更に細分割をする。またオブジェクトを内包しなかったり１つしかない場合はツリーから削除して衝突検出を行わない。

上ではオブジェクトの衝突検出について述べたが、近年では更なる現実性を求めて、オブジェクトの細部に対する衝突や変形するオブジェクトも扱われてきている。例えば、Zhang らは布のリアルタイム・アニメーションをポリゴンレベルの衝突検出を用いて行った [8]。ポリゴンレベルの衝突検出では計算量が先程のオブジェクトレベルとは比較できない程あるため、この研究を含めて大部分の研究では境界ボリュームを用いて計算量の削減を行っている。

2.3.3 境界ボリュームを用いた衝突検出

衝突検出の対象がポリゴンの場合、ポリゴンの数が莫大な為、計算量を減らすために境界ボリューム (bounding volume) が使われる。境界ボリュームは、オブジェクト (ポリゴンの集合) を囲う立体で、アプリケーションの条件や計算機の性能などを考慮して立体の種類を選択する。簡単な境界ボリュームとしては、球 (bounding sphere), 立方体 (bounding box) などがある。特に座標軸に平行な bounding box を AABB(Axis-aligned bounding box, 図 2.2-(a)) という。AABB はオブジェクトの形や向きによって隙間の多い境界ボリュームであるが構造が単純なため境界ボリュームの構築や保持するためのメモリ量が比較的少ない [1]。また、OBB(oriented bounding box, 図 2.2-(b)) はオブジェクトの向きにフィットしたボックスを構築するため AABB と比較して計算量の削減が多くできる反面で、OBB を構築する時間や保持するメモ

リ量が多く必要である。また、更にオブジェクトにフィットする境界ボリュームとしては k-DOP(discrete oriented polytopes, 図 2.2-(c)) があり、これは k 個の任意の方向の面から成る境界ボリュームで数十万ポリゴンで構成されるオブジェクトの衝突検出にも使われている [9]。

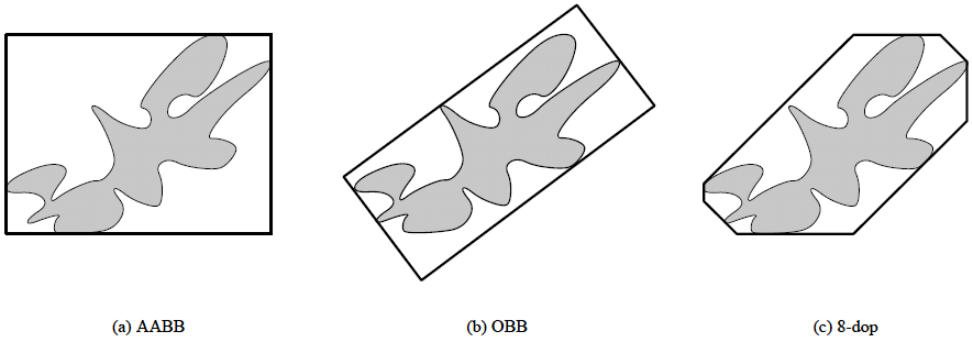


図 2.2: 境界ボリュームの概観 [1]

階層型の境界ボリューム

次に、上で述べた境界ボリュームを階層化した境界ボリュームツリーについて説明する。これは、境界ボリュームを再帰的に細分化する方法で、衝突検出対象のポリゴン数が n 個の場合、ツリーの葉 (leaf) 数が n 個、節 (node) の数は $n-1$ 個のツリーを構築するのが一般的である。この境界ボリューム・ツリーの研究は衝突検出の研究分野で多くの論文が発表されている。Klosowski らは階層型 18-DOP を用いて 1 つのオブジェクトが 143690 ポリゴンで構成された同じオブジェクトのペアに対する衝突検出時間の平均が 0.366ms であった。しかし、この階層型 18-DOP の構築に 217.8sec(約 3 分半) の時間がかかっている [10]。(Silicon Graphics Indigo 195MHz IP28/R10000 processor 192Mbyte)

Bergen[1] は、階層型 AABB を使った衝突検出でオブジェクトの変形時に行うツリーの再構築を高速に行うアルゴリズムを示した。 境界ボリューム・ツリーの再構築時間を OBB ツリー [11] と比較した場合で AABB は OBB の 5%未満の時間で済むことから、変形するオブジェクトの衝突検出への有効性を示した。また、Bergen は衝突検出時間の内訳としてポリゴン・テストが全体に占める時間の割合として AABB ツリーは 28%、OBB ツリーは 5%と、AABB ツリーを使った場合の三角ポリゴンの交差検出処理の高速化の重要性を示した。

2.3.4 距離ベースの衝突検出

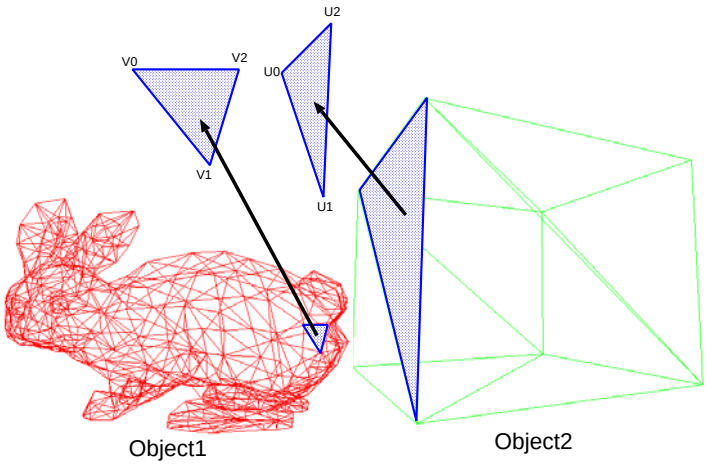
距離ベースの衝突検出に関する研究は、Cohen ら [12] の方法や Gilbert らの GJK 法 [13] がある。これらのアルゴリズムは基本的に、2 つの凸形状オブジェクト間の最近傍点を探索し、当っているか、又は閾値内に近づいているかを求め、当ってない場合は、オブジェクトを引き離すのに必要な方向と最短距離 (貫通深度) を求められる。しかし、衝突検出対象は凸形状のオブジェクトに限られるので、凹形状のオブジェクトは凸形状に分解するか、凸形状の境界ボリュームを構築する必要がある。北村ら [4] は変形するオブジェクトに対する衝突検出では、境界ボリュームと Octree の組合わせたアルゴリズムの方が距離ベースよりも平均的には高速であることを示した。

2.3.5　仮想現実環境における衝突検出

現実に近い環境表現を目指す仮想現実においてリアルタイム性やオブジェクトの変形に対応する事は重要な事である。リアルタイム性を考えると 2.3.3 節の境界ボリュームや 2.3.3 節の階層型境界ボリュームを用いて計算量を減らす事は重要である。しかし、オブジェクトの変形も同時に考えたとき、階層型境界ボリュームは再構築に時間がかかることを 2.3.3 節で述べた。

2.4　三角ポリゴンの交差検出

本論文における衝突検出の対象である三角ポリゴンの” 交差” 検出について説明する。図 2.4 を用いて、２つのオブジェクトに対するポリゴンの交差検出について説明する。Object1 と Object2 は 902 ポリゴンと 12 ポリゴンで構成されている。この２つのオブジェクトに対する三角ポリゴンの交差検出を単純に全探索で行った場合の計算量 (N_p) は $902 * 12 = 10824$ である。



１オブジェクトペアに対する三角ポリゴン交差検出処理の一般的な処理ルーチンを図 4.11 に示す。オブジェクト 1(obj1) とオブジェクト 2(obj2) の各ポリゴン数 (obj1.num, obj2.num) の２重ループになっている。このループ中にある”Tri-Tri_intersect_chk(obj1.T[i], obj2.T[j])“は、obj1 の i 番目の三角ポリゴン (obj1.T[i]) と、obj2 の j 番目の三角ポリゴン (obj2.T[j]]) を入力として、それに対する交差検出処理を行う関数である。この関数の戻り値は交差の有無について bool 値 (true,false) で返される。

2.4.1　三角ポリゴン交差検出アルゴリズム

従来研究で代表的な三角ポリゴンの交差検出アルゴリズムとして、Möller[14] と Held[15] のアルゴリズムがあるが、この両方のアルゴリズムは２つの問題がある。１つは、明示的にいくつかの交差ポイントを構築しなければならない事。もう１つは、１つの値を計算する為の計算回数が多いので浮動小数点数では丸め誤差の問題、固定小数点数では演算回路の増加の問題があった。この問題を大幅に改善したのが、Devillers ら [2] [16] のアルゴリズムでアルゴリズムの安定性 (stability) を 10-20%改善した。

Devillers ら [2] は、１ペアの三角ポリゴン交差検出時間について自分達のアルゴリズムと Möller[14] と Held[15] の計算時間を同じ PC(CPU:PentiumIII 1GHz) で測定した。その結果を 表 2.1 に引用する。この

```
for( i=0; i<obj1.num; i++){
  for( j=0; j<obj2.num; j++){
    Tri-Tri_intersect_chk(obj1.T[i], obj2.T[j]);
  }
}
```

図 2.3: 1 オブジェクトペアに対する三角ポリゴン交差検出ルーチン

結果によると Deviller のアルゴリズムがもっとも早く、交差する場合は 1 ポリゴンペアあたり 343nsec かかることがわかる。次で Devillers のアルゴリズムについて概説する。

表 2.1: 従来研究の処理速度 [2]

Code	Execution times (msec)
交差する場合	
Devilles	0.343
Möller	0.488
Möller (no div)	0.414
Held	0.471
交差しない場合	
Deviller	0.192
Möller	0.248
Möller (no div)	0.229
Held	0.304

2.4.2 Devillers の三角ポリゴン交差検出アルゴリズム

ここでは、三角ポリゴン交差検出ハードウェアに用いる、Devillers[2] の提案したアルゴリズムの概要を図 2.4 を用いて説明する。尚、ここではアルゴリズムの大方なイメージを掴む事が目的なので幾何学的な証明については [2] や、それに書かれている参考文献をたどって頂きたい。尚、私がこのアルゴリズムを理解して実装する上で、このアルゴリズムの C 言語の実装 [17] も参考にした。

Devillers のアルゴリズムを図 2.4 を用いて説明する。このアルゴリズムは大きく次の (1) から (7) の 7 つに分けられる。(1)T_2 の法線ベクトル計算、(2)T_1 と π_2の交差テスト、(3)T_1 の法線ベクトル計算、(4)T_2 と π_1 の交差テスト、(5)T_1 の頂点座標の並び替え、(6)T_2 の頂点座標の並び替え、(7) 交差区間重なりテスト。この内 (1) と (3) は法線ベクトル計算、(2) と (4) はポリゴンと超平面の交差テストで、中でやっている処理内容が同じなので以下で同時に説明する。例として図 2.5 に示す三角ポリゴンペア T_1 と T_2 の交差検出処理についてみてみる。

まず三角ポリゴン T_1 と T_2 の 3 次元頂点座標 $(\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2})$,$(\boldsymbol{U_0}, \boldsymbol{U_1}, \boldsymbol{U_2})$ が (1) に入力される。(1) から (7) までの処理を通して三角ポリゴンの交差判定を行うが、途中 (2) と (4) の時点で交差の可能性が無ければ結

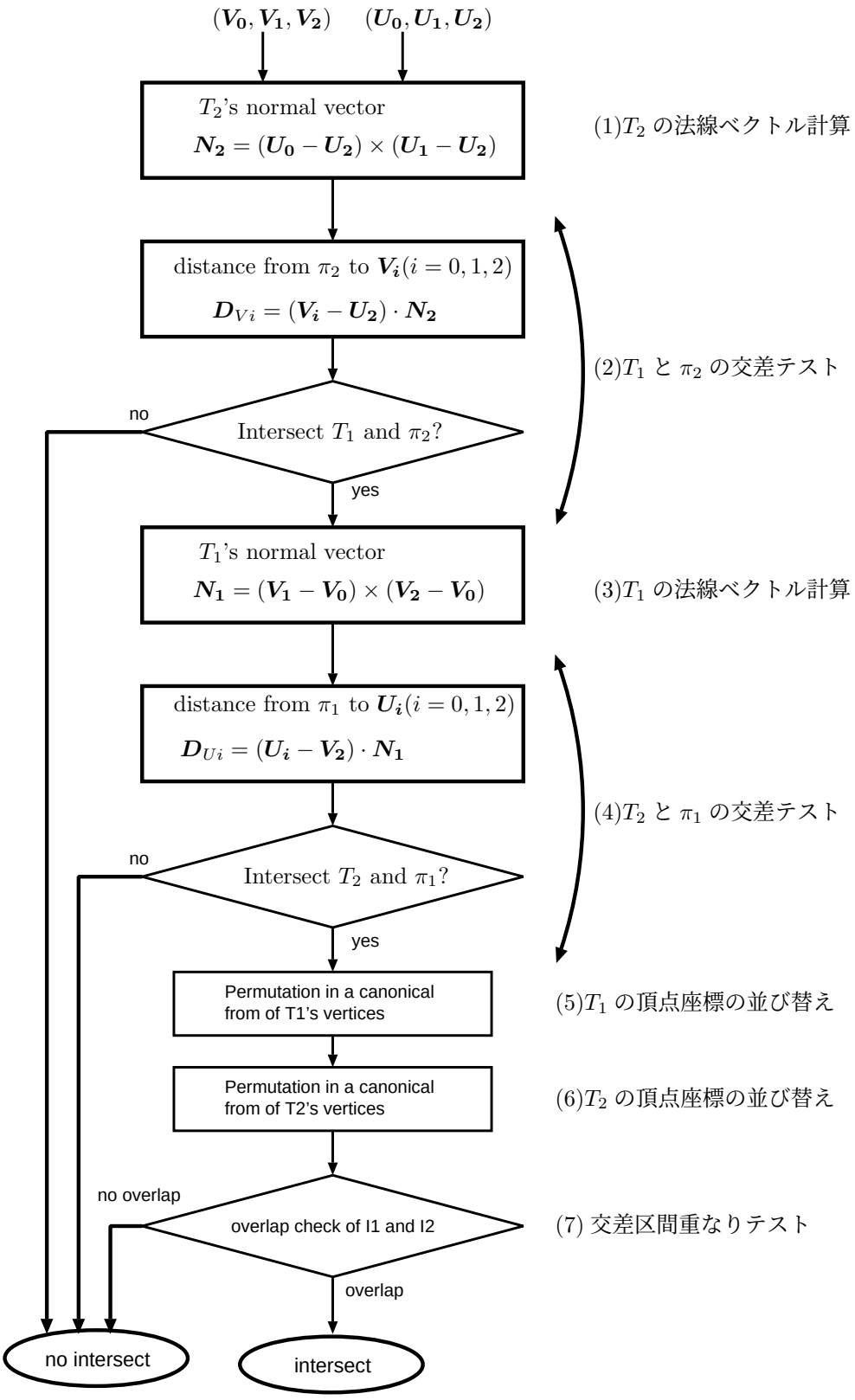


図 2.4: 三角ポリゴン交差検出アルゴリズム

果を出力して終了する。

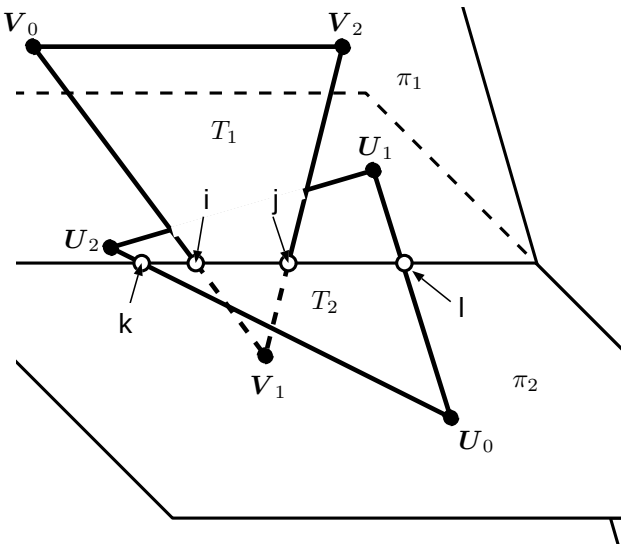


図 2.5: 三角ポリゴン・ペア T_1と T_2

(1),(3) 法線ベクトル計算

(1) を例に説明する。三角ポリゴン T_2(図 2.6) は 3 つ頂点座標 $(\boldsymbol{U_0}, \boldsymbol{U_1}, \boldsymbol{U_2})$ から構成されている。この平面の法線ベクトル (式 2.2)N_2 は右手の法則 (right-hand rule) に基づき頂点が反時計回りに並ぶ時、法線ベクトルは視点を向く。三角ポリゴン T_2 を含む平面 π_2h は式 2.1(X は平面上の任意の点、d_2 は原点から π_2 までの垂直距離) である。Devillers のアルゴリズムでは U_2 を原点として、d_2=0 として計算している。

$$pi_2 : \boldsymbol{N_2} \cdot X + d_2 = 0 \tag{2.1}$$

$$\boldsymbol{N_2} = (\boldsymbol{U_0} - \boldsymbol{U_2}) \times (\boldsymbol{U_1} - \boldsymbol{U_2}) \tag{2.2}$$

(2),(4) $T_1(T_2)$ と $\pi_2(\pi_1)$ の交差テスト

(2) を例に説明する。図**??**は三角ポリゴン T_1 と π_2 の交差の例を示す。この交差判定は、T_1 を構成する頂点 $(\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2})$ の内 1 つだけが π_2 平面を挟んで他の 2 つの頂点の反対側にあれば T_1 と π_2 は交差していることがわかる。図**??**の例では、頂点 $\boldsymbol{V_0}$ と $\boldsymbol{V_2}$ が π_2 平面の上側にあり、$\boldsymbol{V_1}$1 つが π_2 平面の下側にあるので交差している場合である。これを、計算式で判定するには式 2.3 を用いて、T_1 の各頂点 $\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2}$ から π_2 平面までの符合付き垂直距離 (法線方向がプラス)$\boldsymbol{DVi}$ を計算し、条件式 2.4 が成り立つとき (符合が全て同じとき) は T_1 の全ての頂点は π_2 を挟んで同じ側にあるので交差しないことが分かり、その時点で T_1 と T_2 は交差しないことがわかる。逆にそれ以外のとき (符合が 1 つだけ異なるとき) は π_2 を挟んだ両側に頂点があるので交差することがわかる。T_1 と π_2 が交差する場合は図 2.4 の (2) および (4) の判定で"yes" の方向に行き引続き処理を続ける。

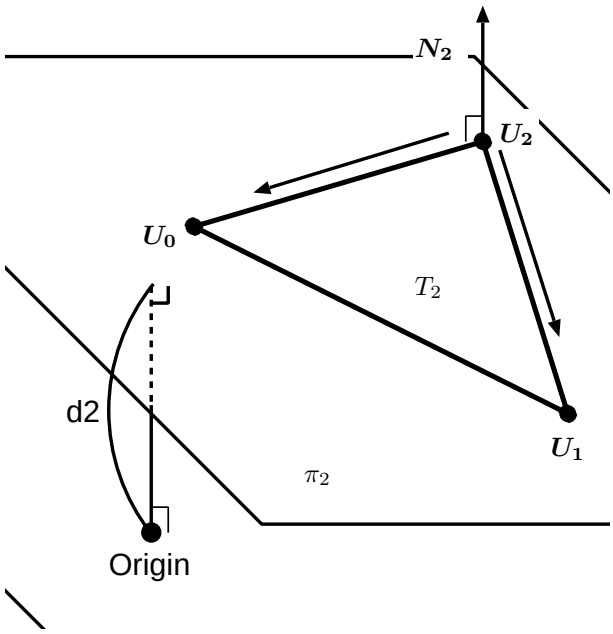


図 2.6: T_2 を含む超平面 π_2

$$\boldsymbol{D}_{Vi} = (\boldsymbol{V_i} - \boldsymbol{U_2}) \cdot \boldsymbol{N_2} \tag{2.3}$$

$$(DV_0 * DV_1 > 0) \wedge (DV_0 * DV_2 > 0) \tag{2.4}$$

(5),(6) T_1 と T_2 の頂点座標の並び替え

(2) と (4) の判定を"yes" で通過した場合、π_1 と π_2 は図 2.5 に示される交差ライン L を共有している。そして、交差直線 L 上の T_1 と π_2、T_2 と π_1 の各交差区間を $I_1 = [i, j]$、$I_2 = [k, l]$ が存在する。しかし、Devillers のアルゴリズムでは交差区間を明示的に計算しないため、右手の法則の性質を用いて、その幾何学的な演算処理で交差判定を行う。具体的には、T_1 と T_2 それぞれ交差ライン L と交差する辺を求める。その為には、まず、ある三角ポリゴンの各頂点について、他方の三角ポリゴンの超平面との関係をみて 1 つだけになる頂点を探す。その頂点からでる両側の辺は交差ライン L と交差することがわかる。ここで具体的に、T_1 と T_2 の関係が図 2.5 の場合を例に説明する。まず、三角ポリゴン T_1 の 3 つの各頂点 T_1 の頂点 $(\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2})$ について、π_2 を挟んで 1 つだけになる頂点を探す。例では、$\boldsymbol{V_1}$ がそれにあたる。ここで頂点の並び順番をこの $\boldsymbol{V_1}$ を先頭にした反時計回りに並び替える。$(\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2}) \Rightarrow (\boldsymbol{V_1}, \boldsymbol{V_2}, \boldsymbol{V_0})$。次に、$\pi_2$ の法線ベクトルの向き (図 2.6) と $\boldsymbol{V_1}$ との関係をみる。π_2 の法線ベクトルは、視線方向を向いているのに対して、$\boldsymbol{V_1}$ は反対方向にあるので、ここで、T_2 の頂点の順番を逆順に並び替える。$(\boldsymbol{U_0}, \boldsymbol{U_1}, \boldsymbol{U_2}) \Rightarrow (\boldsymbol{U_0}, \boldsymbol{U_2}, \boldsymbol{U_1})$。(6) でも同じく、$T_2$ の頂点 $(\boldsymbol{U_0}, \boldsymbol{U_1}, \boldsymbol{U_2})$ と π_1 との間の関係でも上記の処理を行う。

以上の処理を C 言語で説明する。まず、ポリゴン T1 の頂点整列は T1_align(図 2.8) ルーチンで行われる。このルーチンには引数として T_1 と T_2 の頂点座標配列と (2)(3) で計算した $\boldsymbol{DVi}$ と $\boldsymbol{DUi}$ が渡される。そして各頂点の $\boldsymbol{DVi}$ の値が正か負か零で T2_align ルーチンに与えられる引数の順番を変えて頂点整列を行う。

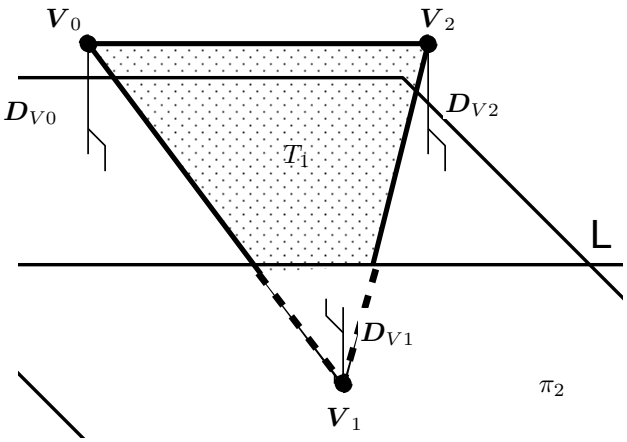


図 2.7: T_1 と π_2 の交差例

次にポリゴン T2 の頂点整列を行う T2_align(図 2.9) ルーチンでは $\boldsymbol{DUi}$ の値を先ほどと同じ様に正か負か零で頂点の順番を整列して次のルーチンである CHECK_MIN_MAX(交差区間重なりテスト) に渡す。

(7) 交差区間重なりテスト

ここでは (5)(6) で整列した頂点座標について条件式 2.5 をチェックし、成り立っていれば、交差区間が重なっているとして結果、三角ポリゴン・ペアは交差していると判定され、条件式が成り立たない場合は三角ポリゴン・ペアは交差しないと判定される。なお、条件式は式 2.6 で定義される。

$$[V_0, V_1, U_2, U1] \leq 0 \wedge [V_0, V_2, U_2, V_0] \leq 0 \tag{2.5}$$

$$[a, b, c, d] := \begin{vmatrix} a_x & b_x & c_x & 1 \\ a_y & b_y & c_y & 1 \\ a_z & b_z & c_z & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} = (d - a) \cdot ((b - a) \times (c - a)) \tag{2.6}$$

2.5 まとめ

本章では、仮想現実環境における衝突検出について定義し、その重要性について書いた。まず、第 2 節では、仮想現実 (バーチャルリアリティ) について現在の状況を概説し、その中でも特に重要な衝突検出技術に注目してその重要性を論じた。第 3 節では、衝突検出について論じた。まず、コンピュータ上でのオブジェクトの表現法について論じ、特に三角ポリゴンに注目してそのデータ構造を明らかにした。そして、衝突検出 (Collision Detection) についてオブジェクトレベルとポリゴンレベルの衝突検出から、特にポリゴンレベルの衝突検出が精密な仮想現実環境の構築に欠かせないことから、三角ポリゴンの交差検出に注目した。そして、仮想現実環境を考えたときオブジェクトを構成する三角ポリゴン数はオブジェクト数と比較に成らない程莫大であるので境界ボリュームを使った計算量の削減法について論じた。さらに階層型の境界ボリュームは非階層型と比較して構築時間がかかる事を従来研究から示し、仮想現実環境ではオブジェクトの変形を考慮するため、非階層型の境界ボリュームが重要であることを論じた。そして、非階層型境界ボ

```
T1_align(V0,V1,V2,U0,U1,U2,dU0,dU1,dU2){
  if (dV0 > 0) {
    if (dV1 > 0) T2_align(V2,V0,V1,U0,U2,U1,dU0,dU2,dU1)
    else if (dV2 > 0) T2_align(V1,V2,V0,U0,U2,U1,dU0,dU2,dU1)
      else T2_align(V0,V1,V2,U0,U1,U2,dU0,dU1,dU2)
  } else if (dV0 < 0) {
      if (dV1 < 0) T2_align(V2,V0,V1,U0,U1,U2,dU0,dU1,dU2)
      else
        if (dV2 < 0) T2_align(V1,V2,V0,U0,U1,U2,dU0,dU1,dU2)
        else T2_align(V0,V1,V2,U0,U2,U1,dU0,dU2,dU1)
    } else {
      if (dV1 < 0) {
        if (dV2 >= 0) T2_align(V1,V2,V0,U0,U2,U1,dU0,dU2,dU1)
        else T2_align(V0,V1,V2,U0,U1,U2,dU0,dU1,dU2)
      }
      else if (dV1 > 0) {
          if (dV2 > 0) T2_align(V0,V1,V2,U0,U2,U1,dU0,dU2,dU1)
          else T2_align(V1,V2,V0,U0,U1,U2,dU0,dU1,dU2)
        }
        else {
          if (dV2 > 0) T2_align(V2,V0,V1,U0,U1,U2,dU0,dU1,dU2)
          else if (dV2 < 0) T2_align(V2,V0,V1,U0,U2,U1,dU0,dU2,dU1)
            else return coplanar_tri_tri(V0,V1,V2,U0,U1,U2,N1,N2);
        }
   }
}
```

図 2.8: ポリゴン T1 の頂点整列コード (C 言語)

```
T2_align(V0,V1,V2,U0,U1,U2,dU0,dU1,dU2) {
  if (dU0 > 0){
     if (dU1 > 0) CHECK_MIN_MAX(V0,V2,V1,U2,U0,U1)
     else
       if (dU2 > 0) CHECK_MIN_MAX(V0,V2,V1,U1,U2,U0)
       else CHECK_MIN_MAX(V0,V1,V2,U0,U1,U2)
     }
     else
       if (dU0 < 0){
         if (dU1 < 0) CHECK_MIN_MAX(V0,V1,V2,U2,U0,U1)
         else
           if (dU2 < 0) CHECK_MIN_MAX(V0,V1,V2,U1,U2,U0)
           else CHECK_MIN_MAX(V0,V2,V1,U0,U1,U2)
       }
       else{
         if (dU1 < 0){
           if (dU2 >= 0) CHECK_MIN_MAX(V0,V2,V1,U1,U2,U0)
           else CHECK_MIN_MAX(V0,V1,V2,U0,U1,U2)
         }
         else
           if (dU1 > 0) {
             if (dU2 > 0) CHECK_MIN_MAX(V0,V2,V1,U0,U1,U2);
             else CHECK_MIN_MAX(V0,V1,V2,U1,U2,U0);
           }
           else{
             if (dU2 > 0) CHECK_MIN_MAX(V0,V1,V2,U2,U0,U1)
             else
               if(dU2 < 0) CHECK_MIN_MAX(V0,V2,V1,U2,U0,U1)
               else return coplanar_tri_tri(V0,V1,V2,U0,U1,U2,N1,N2);
           }
       }
}
```

図 2.9: ポリゴン T2 の頂点整列コード (C 言語)

リュームは階層型と比較して三角ポリゴンの交差検出処理時間が大きいために、この処理の高速化が重要であることを示した。

第3章　三角ポリゴン交差検出ハードウェア

3.1　はじめに

仮想現実環境でより現実環境に近い表現を行うために多数のポリゴンで構成された、多数のオブジェクトを用いる。また、仮想現実環境ではオブジェクト同士の衝突による相互作用が現実的な早さで得られることが求められる。本章ではオブジェクト同士の精密な衝突検出を行うために、オブジェクトを構成する三角ポリゴン同士の交差検出処理を行うハードウェアの構造を示す。、衝突検出システムのボトルネックが三角ポリゴン交差検出処理にあることを示した。そこで本章では三角ポリゴン交差検出処理の高速化を目的とした、専用のハードウェアを検討し、そのハードウェア構成を明らかにする。

本章では、まず最初に、このハードウェアに要求される仕様を従来研究などの文献から洗いだしまとめる。次に、その要求仕様を満たす具体的なハードウェアの構成と工夫点を示し、シミュレーション波形を使ってハードウェアの動作を説明し、最後に本章をまとめる。

1. 要求仕様の検討
2. 要求仕様
3. ハードウェアの構成
4. まとめ

3.2　要求仕様の検討

本ハードウェアの要求仕様の検討項目を以下に示し説明する。

- ハードウェアの実装環境
- ハードウェア化の検討

3.2.1　ソフトウェア環境

ソフトウェアのデータ形式

今回、簡易な仮想現実環境を作成するに辺り、プログラミング言語に C/C++言語 (以降、C 言語) とグラフィックスライブラリに OpenGL を用いた。C 言語を用いてグラフィックスのプログラムを行う時、座標変換を行う為、オーバーフローを気にしなくても良い小数表現を用いた。小数 (実数) タイプの変数には float,double,long double がある。今回、計算精度と速度の中間をとって double 型 (IEEE754 倍精度浮動小数) を選択した。また、三角ポリゴン交差検出アルゴリズムの従来研究 [2, 14] でも double 型を使って実験結果を出している。

入力データサイズ

第２章で示した通り、１ペアの三角ポリゴンの交差検出には６つの３次元頂点座標が必要である。１つの頂点の１次元要素を 1word としたとき１つ三角ポリゴンは 9word 持つことになる。したがって、２ペアの三角ポリゴンでは２倍の 18word の入力が必要である。更に、入力データのサイズを決めるためには 1word のサイズを決定する必要がある。これには入力データの範囲、計算精度、計算方法 (固定小数か浮動小数) の情報が必要である。

3.2.2 ハードウェア環境

ハードウェアの実装環境

本ハードウェアの設計はハードウェア記述言語 (HDL) の１つである VHDL を用いて回路設計を行う。また、ハードウェアの実装を行うデバイスとしては回路の書き換えが自由に行える FPGA(field programmable gate array) というデバイスをターゲットとして、これからの設計を行う。

ハードウェアの計算方式

第２章で説明した通り、三角ポリゴン交差検出処理は計算量が多い。また今回のハードウェアの実装環境である FPGA は ASIC と比較して搭載できる回路規模が小さく、できるだけ回路量を小さくして並列化を行いたい。計算方式として浮動小数点演算と固定小数点演算があるが、浮動小数点演算器は固定小数演算器と比較して回路量が大きくなる上、演算の繰り返しで丸め誤差が交差検出に影響を及ぼす可能性がある。

一方、固定小数点にも数値範囲が狭いという問題があるが、衝突検出は境界ボリュームの重なった範囲内で行われるため、衝突範囲が比較的狭い限りにおいて問題ないと考えられる。

整数化された頂点座標のサイズ

以上に書いたソフトウェア条件で、三角ポリゴンの頂点座標のデータサイズを決める為に次のシミュレーションを行った。シミュレーションは入力データ形式を double 型と多倍長整数 (GMP ライブラリ [18] による) の２種類について行った。小数で表されるポリゴンの頂点座標の整数化については、境界ボリュームの重なる領域のサイズで頂点座標を正規化し、それを $2^{V_{size}}$ することで V_{size} bit の整数値に変換した。この V_{size} bit サイズを決定する為に V_{size} の値を変えて衝突検出判定の幾何学的位置関係についてコンピュータグラフィックスの目視によって正確さの判断を行った。結果、衝突検出の多数の論文で使用されている PLY 形式 [19] の複数のポリゴンデータで、$V_{size} = 23bit$ あれば十分な三角ポリゴンの交差検出精度が得られることがわかった。

尚、ハードウェアの実装経過で、入力を符合つき整数としたので、24bit 目を符合とした $V_{size} = 24bit$ をデータサイズとした。入力は正規化されて正の数になっているので入力時は 24bit 目の符合は無駄であるが今後の課題としたい。

ハードウェアの入力データバス幅

三角ポリゴン交差検出ハードウェアの入力データバス幅 (BUS_{width}) は式 3.1 で表される。

$$BUS_{width} \quad = \quad \text{頂点数} * \text{空間次数} * V_{size} \tag{3.1}$$

本ハードウェアはパイプライン構成で１クロックで１ペアの処理を行いたいので、２つのポリゴンを入力することになる。しかしその場合、入力する頂点数は 6 になるのでデータバス幅 (BUS_{width}) は 432bit 必要である。

ここで三角ポリゴンの交差検出アルゴリズムを図 4.11 で確認すると、交差検出関数に渡すポリゴンの頂点座標データは２重ループで構成されている。そこで、この外側ループのポリゴンを最初に入力してレジスタに記憶しておいて、続いて、内側ループのポリゴンを連続で入力すると一度に入力する頂点数は 3 になるので入力データバス幅は 216bit で済むことになる。

ハードウェアの処理速度

専用ハードウェアを開発するにあたり、まず処理速度の目標を定める。第２章で三角ポリゴンの交差検出アルゴリズムのソフトウェア処理速度を表 2.1 に示した。比較的最近 PC(Pentium3 1GHz) で、１ポリゴンペアあたり 192nsec から 343nsec の処理速度であった。ハードウェアはパイプライン化することでスループットを向上させることが可能で、FPGA では比較的クリティカルパスが長い乗算器専用ブロックでもパイプライン化すれば経験上約 100MHz 程度の動作周波数にできることがわかっているので、ハードウェアの目標処理速度は１ポリゴンあたり 10nsec として、これを目標値にハードウェアのパイプライン化を行う。

ここから、クロックレベルで詳細に考えてみる。3.2.2 節でデータバス幅を 216bit にした。この場合、２つのオブジェクト Obj1 と Obj2 のポリゴン数をそれぞれ N_1 個と N_2 個として、Obj1 を外側ループ、Obj2 を内側ループとしたとき、このポリゴンペアの交差検出にかかるクロックサイクル数 N_{clk} は式 3.2 となる。

$$CLK_{num} = N_1 + N_1 * N_2 + \text{パイプライン段数} \tag{3.2}$$

そして、このポリゴンペアの交差検出にかかる計算時間 T_{send_poly} は、先程の式 3.2 とハードウェアの動作周波数より式 3.3 となる。

$$T_{send\_poly} = \frac{N_{clk}}{\text{動作周波数}} \tag{3.3}$$

3.3 要求仕様

前節で検討した本ハードウェアの要求仕様を以下にまとめる。

- パイプライン 32 段
- ハードウェアの入力データバス幅：216bit
- 動作周波数：100MHz
- 三角ポリゴン交差検出処理速度：約１億ポリゴンペア/sec
- 計算アルゴリズム：Devillers のアルゴリズム

3.4 ハードウェアの構成

図 3.1 に三角ポリゴン交差検出ハードウェアの構成を示す。本ハードウェアは５種類のユニット (法線ベクトル計算、ポリゴン-平面・交差テスト、T1 の頂点座標整列、T2 の頂点座標整列、交差区間重なりテスト) によって構成される。入力として１つの三角ポリゴンを構成する 3 つの 3 次元頂点座標 ($V0_in$, $V1_in$, $V2_in$) が最初のステージである法線ベクトル計算ユニットに入力される。

3.4.1 ポリゴン対超平面交差テストユニットの並列化

前章に記した Devillers らの三角ポリゴン交差検出アルゴリズムのフローチャート図 2.4 と三角ポリゴン交差検出ハードウェアの構成 (図 3.1) を見ながら説明する。ソフトウェアのアルゴリズムでは (1)T_2 の法線ベクトル計算、(2)T_1 と π_2の交差テスト、(3)T_1 の法線ベクトル計算、(4)T_2 と π_1 の交差テストの順番で逐次処理を行た。それに対してハードウェアでは (1)〜(2) と (3)〜(4) を並列に処理を行い、三角ポリゴンと超平面の交差検出結果 (res1(0),res2(0)) を同時に得られる。

3.4.2 ユニットの詳細

法線ベクトル計算ユニット

１つの三角ポリゴン平面の法線ベクトル N の計算を行う。入力は、1 つの三角ポリゴンの頂点座標 ($(\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2})$)、出力は、その三角ポリゴンの法線ベクトル N である。回路の詳細を図 3.2 に示す。１段目は 24bit 対 24bit の三次元ベクトルの引き算を 2 並列で行い、各々25bit の三次元ベクトルを得る。２段目は１段目で得た２つの 25bit の三次元ベクトルを外積計算ユニット (図 3.10) に入力し 6 ステージ掛けて計算を行い、外積 25bit の三次元ベクトル N を出力する。

ベースポリゴンのレジスタへのセット

入力データバス幅については要求仕様の検討でも記述したが１ペアの交差検出には 432bit の入力が必要である。ここで、第１章で述べた１オブジェクトペアの三角ポリゴン交差検出処理図 4.11 を再度みてみる。外ループ (ループ変数 i) の指す三角ポリゴンを１つに対して内ループ (ループ変数 j) のループカウンタ j を obj2.num 回連続でテストしている。内側ループの回転中に１回ずつ外側ループの同じ obj1.T[i] を入力する動作は冗長である。更に、入力直後にある法線ベクトル計算ユニットで行われる計算は、他のポリゴンに左右されないポリゴン固有の計算値であり、これも２つ並列に配置するのはハードウェア領域や電力の無駄である。

従って、本ハードウェアでは法線ベクトル計算ユニット図 3.1 を見ながら、入力データバス幅と回路の削減を行ったハードウェアの説明を行う。入力は InV_0, InV_1, InV_2 で、３つの頂点の３次元座標が入る。まず、本論文では図 4.11 に示される様な外側ループの三角ポリゴンをベースポリゴン (base polygon) とよぶ。ハードウェアで、このベースポリゴンを入力するときは、BSET 信号を'1' にして、法線ベクトル計算ユニットの計算結果 N と頂点座標 (V_0, V_1, V_2) をベースポリゴン専用のレジスタ $(\boldsymbol{V_0(0)}, \boldsymbol{V_1(0)}, \boldsymbol{V_2(0)}, \boldsymbol{N_0})$ にラッチする。ベースポリゴンを入力した次のクロックでは、BSET 信号は'0' となって、内側ループのオブジェクト 2 のポリゴンが入力されて、法線ベクトル計算ユニットの出力を $(U_0(0), U_1(0), U_2(0), N_1N)$ にラッチする。また、ペースポリゴンのレジスタは次に BSET 信号が'1' になるまで値が保持されて次のパイプラインステージのレジスタに渡されていく。

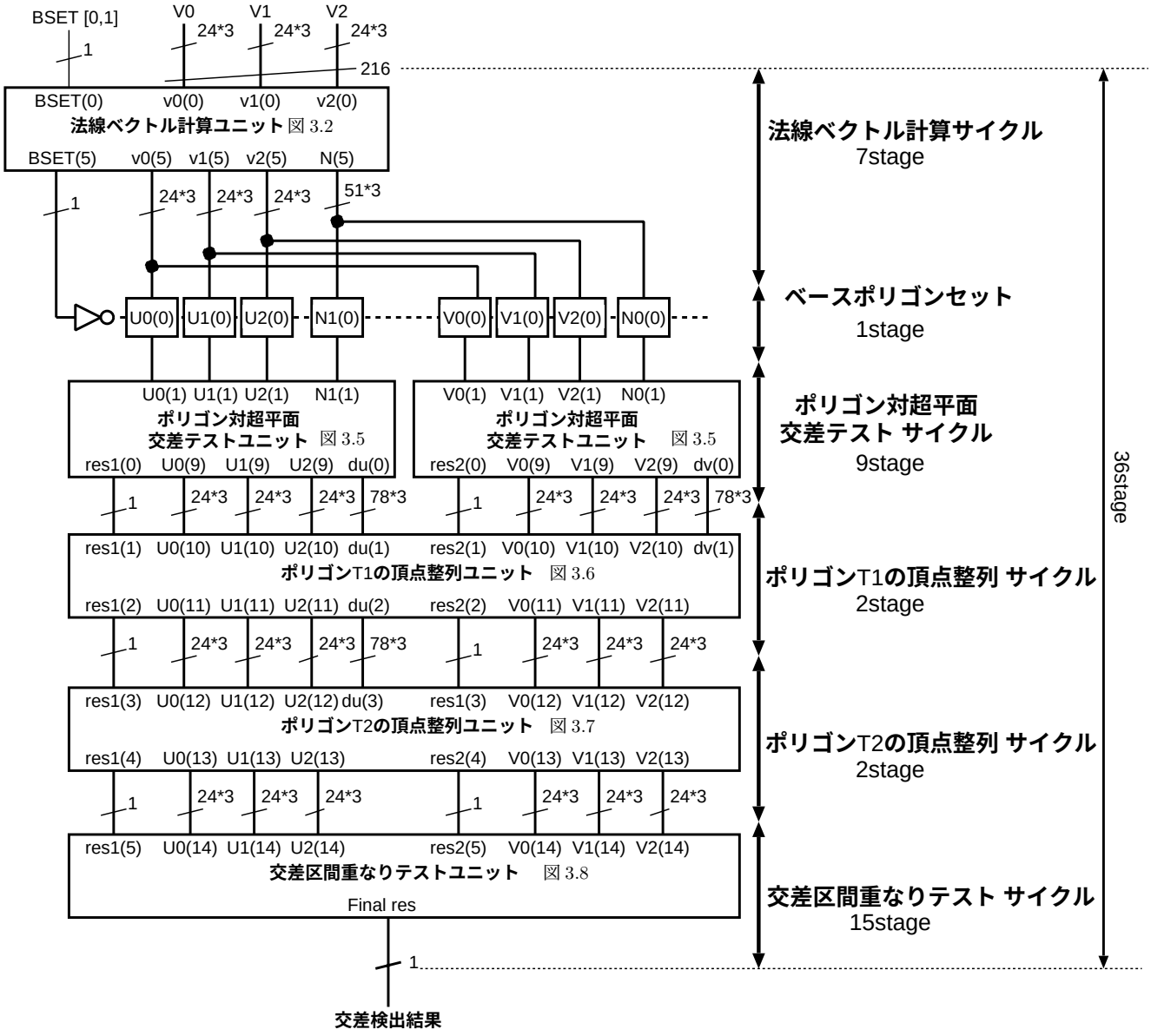


図 3.1: 三角ポリゴン交差検出回路の構成

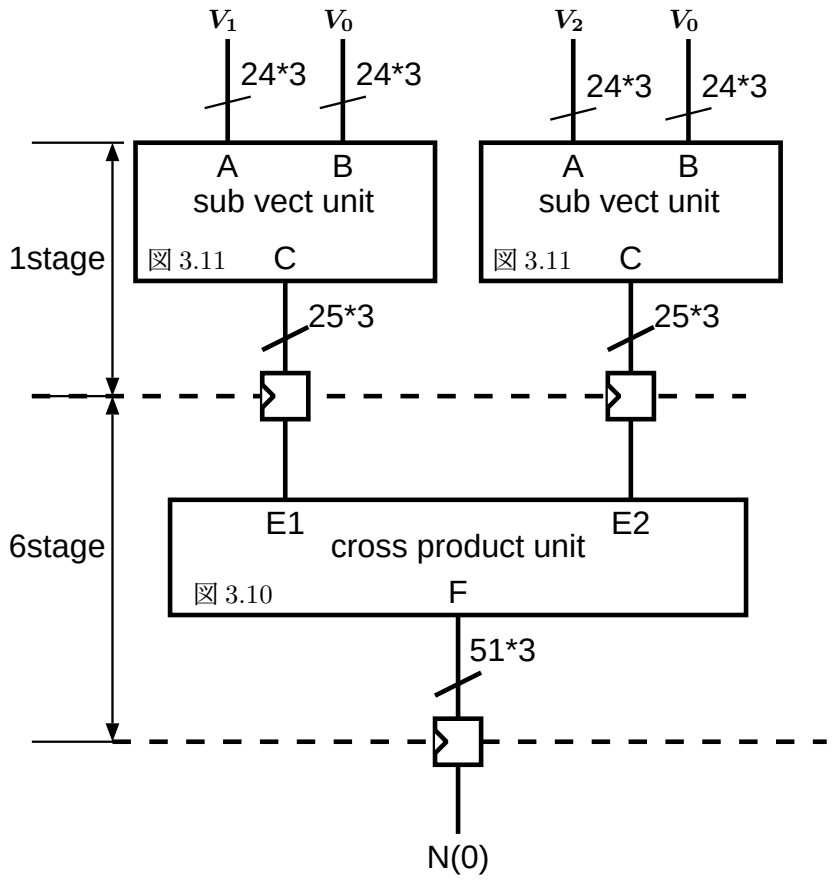


図 3.2: 法線ベクトル計算ユニット

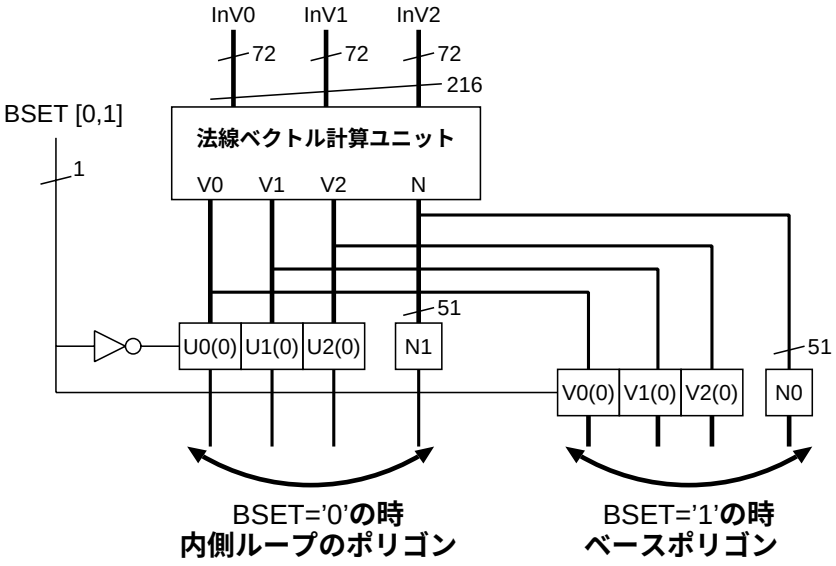


図 3.3: ベースポリゴンのセット

```
process(CLK)
begin
  if(CLK'event and CLK='1') then
    if( BSET='1') then -- BASE polygon のレジスタへのセット
      V0(0) <= IN2_V0;
      N1 <= N;
    else               -- 内側ループの polygon のレジスタへのセット
      U0(0) <= IN2_V0;
      N2 <= N;
  end if;
             :
             :
```

図 3.4: ベースポリゴンのセット (VHDL)

ポリゴン対超平面交差テストユニット

一方の三角ポリゴンから他方の三角ポリゴンによってつくられる超平面 (support plane) に対しての交差の有無をチェックする。入力は、2 つの三角ポリゴンの頂点座標 $(\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2})(\boldsymbol{U_0}, \boldsymbol{U_1}, \boldsymbol{U_2})$、出力は、一方の三角ポリゴンの頂点から他方の三角ポリゴンの超平面までの垂直距離 $(DV0, DV1, DV2)(DU0, DU1, DU2)$ と、ポリゴン対超平面交差テスト結果 (RES0,RES1) である。回路の詳細は図 3.5 に示す。 1 段目は入力が 24bit の三次元ベクトルの引き算を 3 並列で行い、25bit の三次元ベクトルを 3 つ得る。次に 2 段目で先ほどの 3 つの三次元ベクトル各々に対して法線ベクトル N との内積計算を 3 並列で行う。内積計算 (図 3.9) は合計 7 ステージで行われ出力として、3 つの 78bit スカラー値を出力する。内積の次のステージでは、この 3 つのスカラー値の符合が異なっているかチェックする。符合情報は最上位 bit(77bit 目) にある為、これを 2 通り比較して論理和をとれば、 3 つの組合わせの内少なくとも 1 つの符合が異なるか判定できる。

ポリゴン T1 の頂点整列ユニット

三角ポリゴンペア間で向きを合わせる為に T1 の頂点を整列する。入力として、2 つの三角ポリゴンの頂点座標 $(\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2})(\boldsymbol{U_0}, \boldsymbol{U_1}, \boldsymbol{U_2})$、三角ポリゴンの頂点から他方の三角ポリゴンの超平面までの垂直距離 $(DV0, DV1, DV2)(DU0, DU1, DU2)$。出力は、整列した T_1 の頂点座標 $(\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2})$ と、T_2 の頂点座標 $(\boldsymbol{U_0}, \boldsymbol{U_1}, \boldsymbol{U_2})$ と $(DU0, DU1, DU2)$ である。回路の詳細は図 3.6 に示す。 1 段目は $(DV0, DV1, DV2)$ の各々が 0 と等しいか比較を行い結果を bool 値で得る。ま $(DV0, DV1, DV2)$ の各々の符合ビット (23bit 目) だけを SIGN レジスタに入力する。 2 段目は先の SIGN レジスタ値を 2 1 個あるセレクタの各セレクト信号に入力し各出力値を決定する。

ポリゴン T2 の頂点整列ユニット

上記と同様に今度は三角ポリゴン T2 の頂点を整列する。入力として、2 つの三角ポリゴンの頂点座標 $(\boldsymbol{V_0}, \boldsymbol{V_1}, \boldsymbol{V_2})(\boldsymbol{U_0}, \boldsymbol{U_1}, \boldsymbol{U_2})$、三角ポリゴンの頂点から他方の三角ポリゴンの超平面までの垂直距離 $(DU0, DU1, DU2)$。

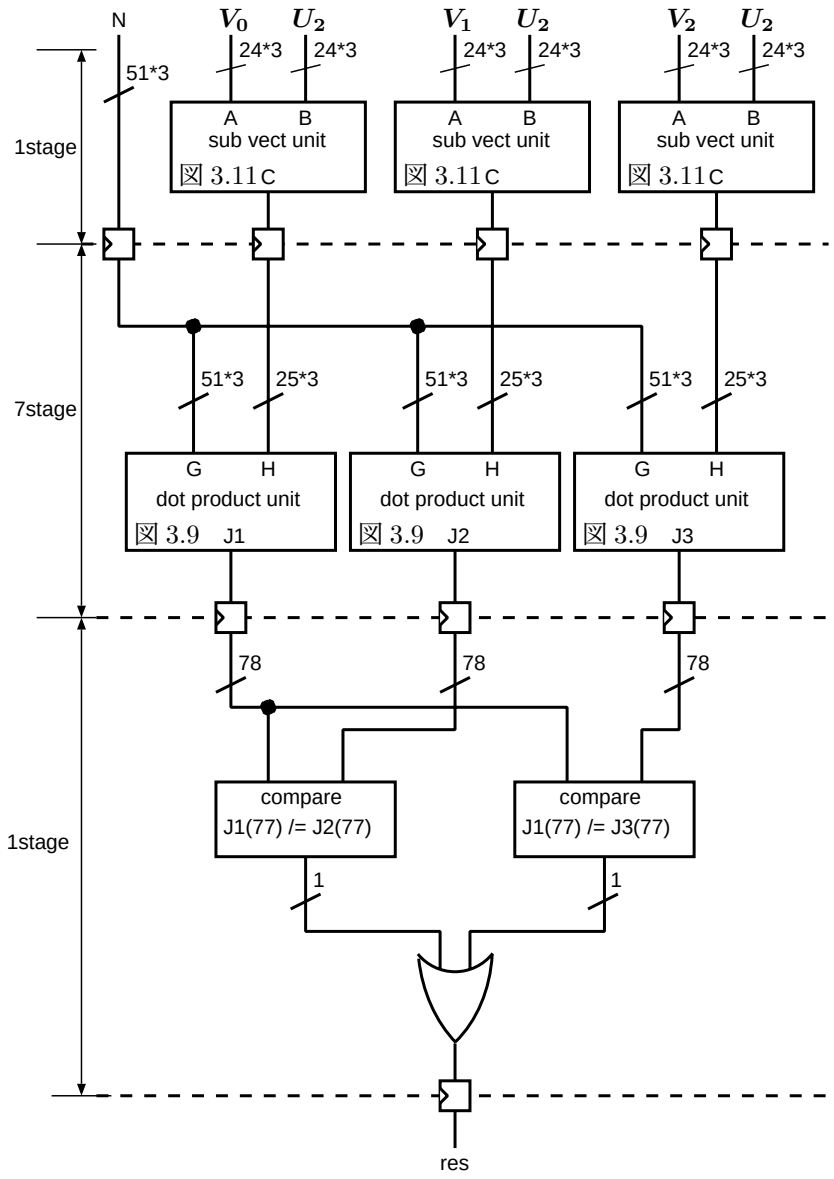
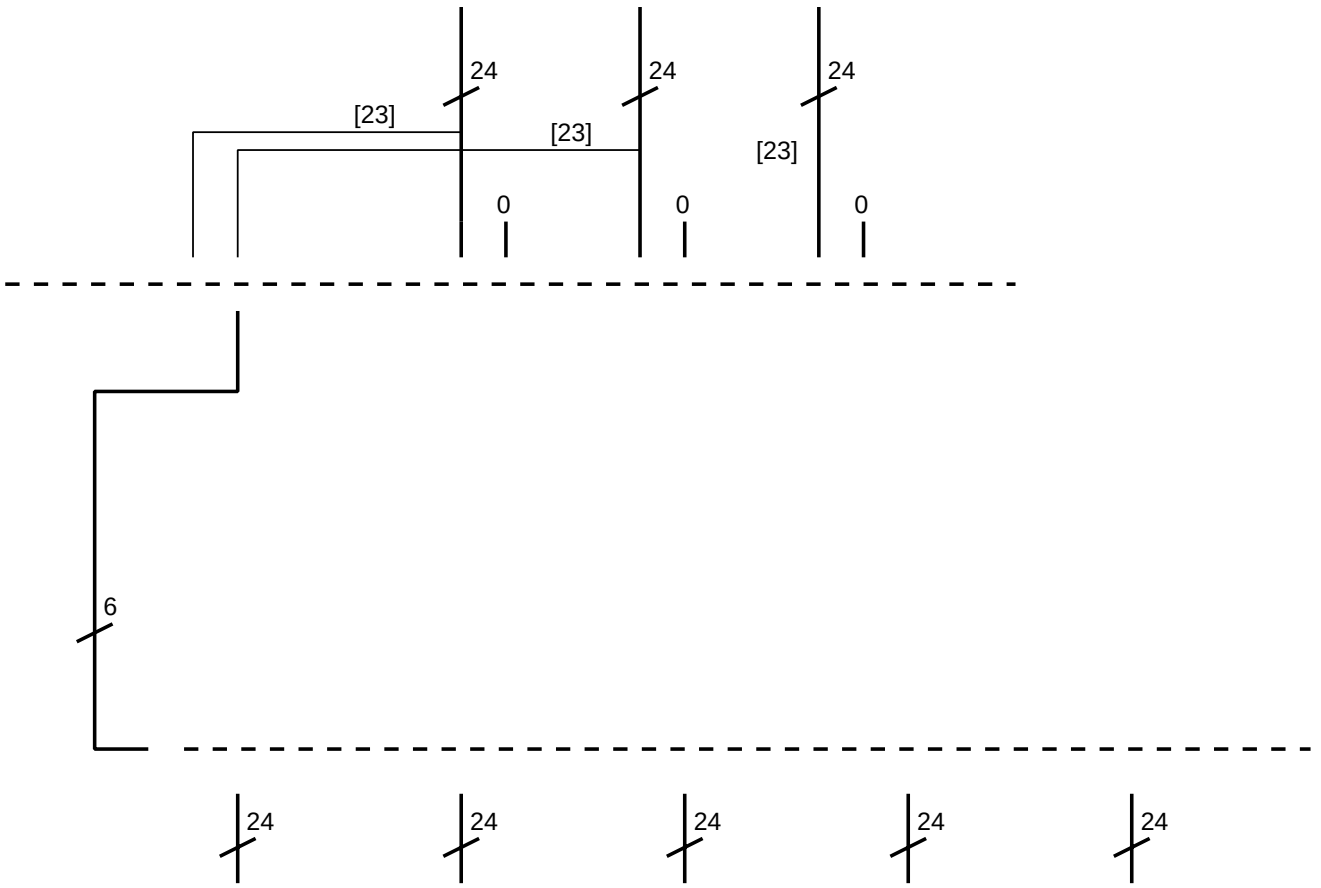


図 3.5: ポリゴン対超平面交差テストユニット


24
24
24
[23]
[23]
[23]
0
0
0
6
24
24
24
24
24