(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4521167号
(P4521167)

(45)発行日　平成22年8月11日(2010.8.11)　(24)登録日　平成22年5月28日(2010.5.28)

(51)Int.Cl.		F I		
G06T 15/40	(2006.01)	G06T	15/40	200
G09G 5/00	(2006.01)	G09G	5/00	550H
G09G 5/377	(2006.01)	G09G	5/36	520M

請求項の数 11　(全 121 頁)

(21)出願番号	特願2003-286311 (P2003-286311)
(22)出願日	平成15年8月4日 (2003.8.4)
(62)分割の表示	特願平9-508604の分割
原出願日	平成8年8月2日 (1996.8.2)
(65)公開番号	特開2004-102998 (P2004-102998A)
(43)公開日	平成16年4月2日 (2004.4.2)
審査請求日	平成15年9月2日 (2003.9.2)
審判番号	不服2007-22809 (P2007-22809/J1)
審判請求日	平成19年8月20日 (2007.8.20)
(31)優先権主張番号	08/511,553
(32)優先日	平成7年8月4日 (1995.8.4)
(33)優先権主張国	米国 (US)
(31)優先権主張番号	08/560,114
(32)優先日	平成7年11月17日 (1995.11.17)
(33)優先権主張国	米国 (US)

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　100147485
弁理士　杉村　憲司
(74)代理人　100072051
弁理士　杉村　興作
(74)代理人　100114292
弁理士　来間　清志
(74)代理人　100134005
弁理士　澤田　達也
(74)代理人　100143568
弁理士　英　貢

最終頁に続く

(54)【発明の名称】グラフィックオブジェクトを画像チャンクにレンダリングして、画像層を表示画像に組み合わせる方法及びシステム

(57)【特許請求の範囲】

【請求項１】

あるビュー空間にフレームレートで画像を生成するシステムにて、画像を生成する方法であって、

現在の画像に対する視体積内のオブジェクトを、少なくとも２つの画像層であるｇスプライトに割り当てるステップと、

第１の３Ｄオブジェクトを第１のｇスプライトにレンダリングするステップを含む、前記オブジェクトを前記少なくとも２つのｇスプライトに個別にレンダリングするステップと、

前記少なくとも２つのｇスプライトを合成し、前記現在の画像を前記フレームレートで生成するステップと、

前記各ステップを繰り返し、後続する画像を前記フレームレートで定まる次の計算期間で生成するステップと、

前記次の計算期間にて、前記第１の３Ｄオブジェクトの動きを近似するとともに、前記第１のｇスプライトを物理的出力装置の座標に変換するための、アフィン変換式を算出するステップと、

前記次の計算期間にて前記第１のｇスプライトを再レンダリングすることよりもレンダリングのオーバーヘッドを低減させるように、前記アフィン変換式を用いて前記第１のｇスプライトに対してアフィン変換を実行するステップとを含む、画像生成方法。

【請求項２】

前記第１のオブジェクトの特徴点を用いて前記アフィン変換式を算出するステップと、
前記アフィン変換式を用いる前記第１のｇスプライトのアフィン変換が、所定の許容範囲内であるか否かについて決定するステップと、
前記アフィン変換が、前記所定の許容範囲内である場合には、前記第１のオブジェクトの動きを近似するように、前記第１のｇスプライトに対してアフィン変換を実行するステップとを更に含む、請求項１に記載の画像生成方法。
【請求項３】
第１の計算期間での前記第１のオブジェクトの特徴点と第２の計算期間での前記第1のオブジェクトの特徴点とを比較するステップと、
前記特徴点における変化が、所定の許容範囲内にないときは、前記第１のオブジェクトを再レンダリングするステップとを含む、請求項１に記載の画像生成方法。
【請求項４】
前記レンダリングするステップが、可変の速度で、前の計算期間においてレンダリングしたｇスプライトを更新するステップを含む請求項１に記載の方法。
【請求項５】
画像層であるｇスプライトを用いて、あるビュー空間にフレームレートで表示画像を生成する画像処理システムであって、
ｇスプライトメモリと、
オブジェクト及び視点の位置を記述している入力値を受信する画像前処理装置であって、視体積と交差する前記オブジェクトの位置を決定し、現在の画像に対する前記視体積と交差するオブジェクトを少なくとも２つのｇスプライトに割り当て、第１の３Ｄオブジェクトの動きを近似するとともに、該第１の３Ｄオブジェクトがレンダリングされた第１のｇスプライトを物理的出力装置の座標に変換するのに用いられるアフィン変換式を算出する、前記画像前処理装置と、
前記画像前処理装置と結合する画像処理装置であって、前記オブジェクトを前記少なくとも２つのｇスプライトに個別にレンダリングし、該レンダリングは前記第１の３Ｄオブジェクトを前記第１のｇスプライトにレンダリングすることを含み、前記少なくとも２つのｇスプライトを前記ｇスプライトメモリに格納し、前記ｇスプライトメモリから前記少なくとも２つのｇスプライトを読み取り、前記アフィン変換式に従って前記第１のｇスプライトを物理的出力装置の座標に変換し、前記物理的出力装置に表示するために、当該少なくとも２つのレンダリングしたｇスプライトを合成し、前記現在の画像を前記フレームレートで生成する前記画像処理装置と、を備え、
前記画像前処理装置は、
前記少なくとも２つのレンダリングしたｇスプライトを合成後、後続する画像を前記フレームレートで定まる次の計算期間で生成するために前記アフィン変換式を算出する処理を繰り返し、前記次の計算期間にて、前記第１の３Ｄオブジェクトの動きを近似するとともに、前記第１のｇスプライトを物理的出力装置の座標に変換するための、アフィン変換式を算出するように構成され、
前記画像処理装置は、
前記次の計算期間にて前記第１のｇスプライトを再レンダリングすることよりもレンダリングのオーバーヘッドを低減させるように、前記アフィン変換式を用いて前記第１のｇスプライトに対してアフィン変換を実行するように構成されている、画像処理システム。
【請求項６】
前記画像前処理装置は、コンピュータシステムにおけるプログラム可能なプロセッサを備える、請求項５に記載の画像処理システム。
【請求項７】
前記画像前処理装置は、プログラム可能なディジタル信号プロセッサを備える、請求項５に記載の画像処理システム。
【請求項８】
前記画像前処理装置は、コンピュータシステムにおけるプログラム可能なプロセッサと

、前記プログラム可能なプロセッサと結合する、プログラム可能なディジタル信号プロセッサとを備える、請求項５に記載の画像処理システム。
【請求項９】
前記画像処理装置は、前記オブジェクトを前記ｇスプライトにレンダリングし、且つ、前記ｇスプライトを前記ｇスプライトメモリに格納するタイラーを有する、請求項５に記載の画像処理システム。
【請求項１０】
前記画像処理装置は、前記ｇスプライトメモリから前記ｇスプライトを読み取り、且つ、前記ｇスプライトを前記物理的出力装置の座標に変換するｇスプライト・エンジンを有する、請求項５に記載の画像処理システム。
【請求項１１】
前記画像処理装置は、前記物理的出力装置に表示するために、ｇスプライトを合成する前記ｇスプライト・エンジンに結合される合成バッファを有する、請求項１０に記載の画像処理システム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、包括的にはグラフィックスレンダリングに関し、特に、改良された、グラフィックオブジェクトをレンダリングする方法及びシステムに関する。
【背景技術】
【０００２】
近代生活のあらゆる分野にコンピュータが普及するにつれて、視覚情報を用いてマン・マシン・インターフェイスを改善しようという要求が増している。グラフィック・ソフト及びハードウェアの進歩は、すでにマン・マシン・インターフェイスを急速に改善してきた。例えば、デスクトップ・コンピュータ用のウィンドウ環境のようなインタラクティブ・グラフィックスは、コンピュータの使い勝手とユーザとの相互作用を大きく改善し、今日では当たり前のことになっている。ハードウェアのコストパフォーマンスが改善するにつれて、コンピュータ上で作成されたグラフィックスやアニメの使用がさらに浸透してきている。残念ながら、真にインタラクティブな、リアルな効果を生み出そうとすると、そのコストのために、応用に限界がある。従って、より低いコストで、さらにインタラクティブで、リアルな効果を与えることのできる新しい画像処理技術やアーキテクチャに対しては需要がある。
【０００３】
画像処理法の分類法には各種あるが、よくある方法の１つは、画像を、それが表わしたいオブジェクトの次元で描出することである。例えば、あるグラフィックスシステムは、オブジェクトを二次元（例えば、ｘ座標とｙ座標を持つ)で表わす。この場合は、そのグラフィックスは「二次元性」(２Ｄ)であるという。また、三次元(例えば、ｘ 、ｙ 、及び、ｚ 座標を持つ)で表わせば、その場合は、そのグラフィックスは、「三次元性」(３Ｄ)であるという。
【０００４】
陰極線管(ＣＲＴ) のような表示装置は、二次元性(２Ｄ)であり、コンピュータのグラフィックスシステムによって表示される画像は一般に２Ｄである。しかしながら、以下にさらに詳しく論ずるように、コンピュータが、三次元空間で画像化されたオブジェクトを表わすグラフィック・モデルを保持している場合、コンピュータは、その表示画像を変えて、３Ｄ空間において同じオブジェクトについて種々の遠近像を描くことができる。一方、２Ｄグラフィック画像は、表示の前に変換する（例えば、 拡大・縮小、変換、回転）ことはできるが、コンピュータは、３Ｄ空間における種々の遠近像から、そのオブジェクトの画像表現を容易に描写することはできない。
【０００５】
現代のコンピュータが、２Ｄグラフィックス、特に３Ｄグラフィックスを次第に効率的

に扱うことができるようになったために、様々の応用分野が広がっており、同時に、コンピュータとユーザ間のインターフェイス(ＵＩ)に基本的な変化が生じている。３Ｄグラッフィクスの利用可能性は益々重要になってきている。即ち、生産品質のアニメ・フィルムの制作ツールを始め、家庭向きの低解像度のゲームやマルチメディア製品を含めた、娯楽関連性の応用分野が成長しているからである。３Ｄグラフィックスの関連する他分野について幾つか挙げるならば、教育、テレビ会議、ビデオ編集、インタラクティブ・ユーザ・インターフェイス、コンピュータ支援設計やコンピュータ支援製造（ＣＡＤ ／ＣＡＭ）、科学・医学画像法、ビジネス応用、及び、電子出版がある。

【０００６】

グラフィックス処理システムは、アプリケーション・モデル、アプリケーション・プログラム、グラフィックス・サブシステムを始め、コンピュータの通常のハードウェア、ソフトウェア部分、及び、付属品を含むと考えてよい。

アプリケーション・モデルとは、もちろん画像処理はモデルに基づくと仮定して、データ、即ち、表示されるオブジェクトを表わす。モデルは、オブジェクトの形を規定するプリミティブ、例えば、点、線やポリゴンを始め、オブジェクトの属性（例えば、カラー）に関する情報を含む。アプリケーション・プログラムは、アプリケーション・モデルへ入る入力、モデルから出て行く出力を制御し、アプリケーション・モデルと、グラフィックス・サブシステムとの間の変換手段として有効に機能する。最終的には、グラフィックス・サブシステムは、ユーザ入力をアプリケーション・モデルに送り、また、アプリケーション・モデルによって保持されるディテール情報に基づいて画像を生成する責任を負う。

【０００７】

通常のグラフィックス処理システムは、物理的出力装置を備える。これが、画像の出力、即ち、表示を行なう。従来他の形態の表示装置も開発されてはきたが、今日優勢な技術は、ラスター・グラフィックスに依拠している。ラスター・グラフィックス装置は、一連の不連続な点、即ち、画像要素（即ち、画素）を含む。画素は、複数の列と行に構成されて、画像を生成する。ＣＲＴ においては、この画素は、ＣＲＴのガラス・画面上に与えられた蛍光体アレイに対応する。このアレイの中の各蛍光体からの光の放射は、それぞれ独立に、このアレイを順々に「走査」する電子ビームによって制御される。このアレイの奇数・偶数列のインターリーブ走査も、例えば、テレビジョン環境では普通に見られる技法である。画面にマップされる画素値のアレイは、ビットマップ、又は、ピックスマップと呼ばれることが多い。

【０００８】

ラスター・グラフィックス装置に関連する１つの問題は、たった１つの画像に対するビットマップでも、それを保存するのに必要なメモリが膨大になることである。例えば、１２８０×１０２４ （即ち、画素行と画素列の数）の表示解像度を維持するのに、この装置は、ランダム・アクセス・メモリの３．７５メガバイト(Ｍｂ)を必要とすることがある。この情報は、繰り返すが、単一画面の画像を表わすにすぎないものであり、フレームバッファと呼ばれる、コンピュータの表示記憶装置の一部に蓄えられる。

ＣＲＴ のような従来のラスター・グラフィックス装置に関するもう１つの問題は、この装置から発せられる光が比較的速やかに減衰することである。そのため、画像の「フリッカ」を避けるためには、通常、６０Ｈｚ以上もの表示速度で、「リフレッシュ」してやらなければならない（すなわち、ラスターの再走査）。このため、グラフィックス処理システムは、画像データを一定速度で供給するという厳しい要求をつきつけられることになる。２つのフレームバッファを用いてこの問題に対応しようとする幾つかのシステムがある。即ち、フレームバッファのうちの一方は、次の画像フレームに対応するピックスマップ情報で更新され、他方のバッファは、現在の画像フレームのピックスマップで画面を更新するのに用いられる。

【０００９】

グラフィックス処理システムに対する要求は、情報の複雑さによってさらに悪化される。そのような情報は、アプリケーション・モデルによって保持されるオブジェクトからの

画像をレンダリングするように、幾度も処理しなければならないからである。例えば、三次元サーフェイスモデルそれ自体が、複雑なタスクとなる。サーフェイスモデルは、アプリケーション・モデルによって実行され、ポリゴンメッシュ、パラメトリック曲面、又は、方形面が用いられる。平面ポリゴンのメッシュによって、曲面を表現できるが、レンダリング画像におけるその画像表現の「滑らかさ」は、表示面の解像度、及び、サーフェイスモデルに用いられた個々のポリゴンの数、の双方に依存する。ポリゴンメッシュに基づく複雑な面についての高解像度のモデル化に伴って、そのサーフェイスモデルの複雑な計算は、リソースを極端に集約させる。

【００１０】

上に紹介したように、よりリアルな、よりインタラクティブな画像を生成したいという要求がある。「リアルタイム」という用語は、通常、インタラクティブで、リアルな画像処理システムを云う場合に用いられる。「リアルタイム」装置においては、ユーザは、ある場面内のオブジェクトの連続的な動きを感ずることができなければならない。リアルタイム機能を持つビデオゲームにおいては、活動的キャラクタと視点は、ユーザの入力に対して極小の遅延で反応し、かつ、滑らかに動く必要がある。

【００１１】

そのようなリアルタイム効果を実現するためには、画像レンダリングシステムは、十分に高速で新しい画像を生成し、それによって、ユーザが、ある場面内のオブジェクトの連続的な動きを感知できるほどになっていなければならない。新たな画像を表示するために計算される速度は、“計算”レート又は“計算フレーム”レートと称する。リアル効果を実現するために必要とされる計算レートは、オブジェクトが場面内でどのくらいの速さで動き回るか、及び、遠近像の視点がどのくらい急速に変化するか、によって変わる。通常のアプリケーションでは、リアルタイムグラフィックスシステムは、新画像を１秒当たり少なくとも１２個計算し、それによって、連続する動きをシミュレートする一連の画像を生成する。高品質アニメ・アプリケーションでは、計算レートは、それよりもはるかに高くなければならない。

【００１２】

リアルタイム装置に関するもう１つの問題は、伝達遅延である。伝達遅延とは、ユーザからの入力、即ち、場面内のあるキャラクタを動かすためのジョイスティックの動きに対して画像を計算して表示するのに必要な時間である。伝達遅延時間がユーザにとって目立つものであればあるほど、その程度に応じて、「リアルタイム」の相互作用性は損なわれる。理想的には、ユーザに、伝達遅延を気づかせてはならない。しかしながら、実際には、新しい入力に対して、ある場面のオブジェクトをレンダリングし表示画像を生成するのに必要な、幾らかの遅延が必ず存在する。リアルタイム相互作用性の改善は、データを切り落とすことなく達成できれば極めて望ましい。なぜなら、データの切り落としは、画像品質を損なうからである。

【００１３】

始めに紹介したように、通常のグラフィックスシステムは、一般に、フレームバッファを備える。画像を生成するためには、グラフィックスシステムは、ある場面内のオブジェクトの全てをレンダリングし、得られた画像をこのフレームバッファに蓄える。次に、その装置は、レンダリングした画像データを表示装置に転送する。通常のグラフィックス・アーキテクチャであれば、フレームバッファの全内容は消去され、場面は再レンダリングされ、新たなフレーム画像を生成する。このタイプの装置では、各オブジェクトは、フレームごとに再レンダリングされる。なぜなら、フレームバッファは、フレームとフレームの間において空にされるためである。従って、各オブジェクトは、同じ速度で最新画像に更新される。これは、その画像が、その場面内で現実にどのような動きをしているか、又は、特定の用途に対して、それがどのような重要性持つか、には無関係に行なわれる。

【００１４】

この通常のアーキテクチャは、高度にリアルで、インタラクティブなグラフィックスを生成しようとすると、幾つかの障害に直面する。第一に、ある特定のフレームに関して、

ある場面内の各オブジェクトが全て同じ優先性で、同じ更新速度でレンダリングされる。従って、 ほとんどディテールを持たず、動いてもいない背景のオブジェクトも、より早く動き、より細かい面ディテールを持つ前景のオブジェクトと同じ速度で再レンダリングされる。そのため、その処理や、その記憶のためのリソースが、背景オブジェクトの再レンダリングのために消費されることになる。しかも、これら背景オブジェクトは、フレームごとにほどんど大した変化をしないのに、である。

【００１５】

この従来のアーキテクチャに関するもう１つの欠点は、場面における各オブジェクトが全て同じ解像度でレンダリングされるということである。その結果、この種のやり方で消費されるレンダリング・リソースは、そのオブジェクトが、場面全体に対して占める重要性ではなく、そのオブジェクトが占める画面面積の大きさに比例することになる。この問題を明らかにするには実例が役立つ。典型的なビデオゲームの場合、前景の、フレームごとに変化する活動的なキャラクタと、フレームごとにほとんど変化しない背景とがある。メモリ消費という点でのコストは、背景を生成する方が、活動的なキャラクタを生成するのに要するものよりもはるかに高い。なぜならば、背景の方が、画面上において、より大きな面積を占めるからである。画像データは、背景オブジェクトが占める各画素位置について保存されていなければならない。一方、より小さい、活動的なキャラクタの方は、画素データは、小型のキャラクタの占める画素に対してだけ、生成・保存される。その結果、背景は、場面内の重要性はずっと低いにも拘わらず、より大きなメモリを占有する。さらに、通常のアーキテクチャでは、背景全体が、各フレームに再レンダリングされ、貴重な処理用リソースを消費することになる。

フレームバッファ法の主要利点は、その出力装置の空間的解像度及び輝度解像度によってしか制限されないという条件で、任意の数のプリミティブオブジェクトを、物理的出力装置において、１つの任意の画像を構成するように用いることができることである。しかしながら、フレームバッファを用いるグラフィックシステムには幾つかの弱点がある。

【００１６】

フレームバッファは、高価なメモリを大量（例えば、６４－１２８Ｍｂ）に使う。標準のランダム・アクセス・メモリ(ＲＡＭ) では、アクセス・スピードが遅いので、フレームバッファには適しない。例えば、１０２４×１０２４ 画面において、万の桁の画素を空にするには、各メモリ・サイクルが２５０ナノ秒を要すると仮定して、１／４秒かかる。従って、さらに高速の、さらに高価な、ビデオＲＡＭ(ＶＲＡＭ)、又は、ダイナミックＲＡＭ(ＤＲＡＭ) が、通常、フレームバッファには使われる。高性能装置ともなれば、２個の高価なフレームバッファを備えているものも多い。即ち、一方のフレームバッファは、現在の画面を表示するのに用いられ、他方は、次のフレームをレンダリングるのに用いられる。この大容量の専用メモリは、グラフィックススシステムのコストを大幅に増大させる。

【００１７】

フレームバッファのメモリバンド幅がさらに問題である。各画素に蓄えられたテクスチャ（texture）、カラー、及び、深度の情報によって画像処理を実行するのをサポートするためには、３０Ｈｚで画像処理するとして、約１．７ ギガバイト／秒のバンド幅が必要である。通常のＤＲＡＭは、僅か５０Ｍｂ／秒しか持っていないので、所望のバンド幅が得られるように、フレームバッファを並行処理法で処理される多くのＤＲＡＭから構成させなければならない。

【００１８】

リアルタイムな、インタラクティブな効果を実現するために、高品質グラフィックスシステムは、並行レンダリングエンジンを用いる。大きなフレームバッファに関わる問題に対処するために、従来、３つの基本的な並行ストラテジが開発されている。(１) レンダリング処理を複数のプロセッサに分配するパイプライン方式、(２) フレームバッファ・メモリを、各々独自のプロセッサを備えたメモリ・チップ・グループに分割する分割方式、及び、(３) フレームバッファ・メモリ・チップ上の処理回路と高密度メモリ回路とを

組み合わせる組み合わせ方式、である。これらの方式は、大型フレームバッファを用いるグラフィックスシステムの処理法を改善したが、このシステムのコストをも飛躍的に増大させた。

【００１９】

高価な並行処理法を用いてさえ、高度なアンチエイリアシング法をサポートするのは極めて難しい。アンチエイリアシング法というのは、連続面を不連続な画素で表わすことによって生じる、レンダリング画像におけるアーティファクトを減少させるための処理を云う。通常の、フレームバッファ・アーキテクチャでは、ある全体フレームにおける画素値は、任意の順序で計算される。従って、高度なアンチエイリアシングを実行するためには、アンチエイリアシングを始める前に、全フレームの画素データが生成されていなければならない。リアルタイム装置においては、画素データに対してアンチエイリアシングを実行するのに十分な時間が残されていない。処理を実行すれば必ず伝達遅延が更に加わることになる。更に、アンチエイリアシングは、画素フラグメントを保存するためのメモリをさらに必要とする。フレームバッファは、すでに大容量の高価なメモリを含んでいるから、アンチエイリアシングをサポートするために、更に専用メモリを加えることは、フレームバッファシステムを更に一段と高価なものにする。

【００２０】

画像圧縮法も、画像処理にフレームバッファを用いるグラフィックスシステムにおいては、容易に使用できるものではない。フレームバッファによるグラフィックスシステムにおいて、処理速度を上げるために用いられる並行処理法は、圧縮法を導入しようという試みに幾つかの障害をもたらす。並行処理中、フレームバッファのどの部分に対しても、どの時点においても、ランダムにアクセスすることができる。一方、多くの画像圧縮法では、画像データは、圧縮処理中は変化しないことを要求する。それによって、後になって、画像データを復元できるようにするためである。

【００２１】

フレームバッファ・アーキテクチャでは、高価なメモリや並行処理用ハードウェアは、常にあまり使用されずに放置されがちであった。なぜなら、ある任意の時点においては、フレームバッファ・メモリ、又は、並行処理ユニットのごく小部分のみが実際に使用されるにすぎないからである。従って、フレームバッファ・アーキテクチャは、大量の高価なメモリや処理用ハードウェアを備えているが、このハードウェアは十分に活用されていない。

【発明の開示】

【課題を解決するための手段】

【００２２】

発明の概要

本発明は、幾何形状プリミティブのようなグラフィック・データをレンダリングして、表示画像を生成するための方法、及び、システムを提供する。本発明は、３Ｄグラフィックスをリアルタイムでレンダリングするのに特に好適であるが、他のグラフィックスや画像処理アプリケーションにも同様に適用できる。

グラフィックス・レンダリングシステムの１つの態様として、本システムは、グラッフィクオブジェクトを、ｇスプライト(gsprite) と呼ばれる画像層に別々にレンダリングし、次に、このｇスプライトを合成して表示画像に仕立てあげる。さらに具体的に言うと、本システムは、ｇスプライトをオブジェクトに割り当て、次に、各オブジェクト(単数又は複数)を、対応するｇスプライトにレンダリングする。ｇスプライトをレンダリングするために、本システムは、画像領域、即ち、ｇスプライトのチャンクを連続的にレンダリングする。本システムは、ｇスプライトを、チャンクに分割し、これらのチャンクの間でオブジェクトの大きさをソートし、次に、そのチャンクを連続的にレンダリングする。本システムは、ｇスプライトを合成して表示画像に仕立てあげる。

【００２３】

本発明の一態様として、３Ｄオブジェクトの動きをシミュレートするように、及び、レ

ンダリング・オーバーヘッドを低減させるように、ｇスプライトを変換させることができる。一実現例では、本システムは、ある場面内のオブジェクトを、別々のｇスプライトにレンダリングする。あるオブジェクトをあるｇスプライトにレンダリングした後、本システムは、そのオブジェクトを再レンダリングする代わりに、次のフレーム用のｇスプライトを再利用する。これを実行するために、本システムは、ｇスプライトの表わす３Ｄオブジェクトの動きをシミュレートするアフィン変換式を算出する。本システムは、ｇスプライト上でアフィン変換を実行し、このｇスプライトと他のｇスプライトとを合成し、表示画像を生成する。

【００２４】

本発明の別の態様として、本システムは、画像データのチャンクのために画素フラグメントを処理する手法にある。本システムは、チャンク用のプリミティブをラスタライズし、全体的に、又は、部分的にプリミティブでカバーされる画素位置用の画素データを生成する。あるプリミティブが、画素位置を部分的にカバーする場合、又は、半透明性である場合、本システムは、画素フラグメントを生成し、この画素フラグメントをフラグメントバッファに蓄える。あるプリミティブが、画素位置を完全にカバーし、かつ、不透明性である場合、本システムは、そのカラーデータを画素バッファに蓄える。本システムは、チャンク用のプリミティブをラスタライズし、次に、後処理ステップにおいてチャンク用の画素データを解析する。チャンクをレンダリングするアーキテクチャは、一方では、表示画像をリアルタイム速度で生成しながら、他方では、高度なアンチエイリアシングをこの画素データに対して実行することを可能にする。

【００２５】

本発明の別の態様は、システム内のラスタライザが、生成された画素フラグメントを、フラグメントバッファに蓄えられたフラグメントとマージすることを試みることによって、フラグメント記録を保存できるようにする手法にある。蓄えられているフラグメントが、生成されたフラグメントの所定の深度及びカラー値の許容範囲内にあれば、本システムの画素エンジンは、これらフラグメントをマージする。画素エンジンは、これらのフラグメントを、一般化フラグメントと蓄えられたフラグメントのカバレッジデータ(例えば、カバレッジマスク)を組み合わせることによって部分的にマージする。マージされた画素フラグメントが、完全にカバーされ、且つ、不透明性であれば、画素エンジンは、その画素フラグメントを対応する画素バッファ・エントリに移し、フラグメント記録をフラグメントメモリから解放する。

【００２６】

本発明のさらに別の態様は、サブシステムが、フラグメント記録のリストを解析する手法にある。一手法として、フラグメント解析サブシステムは、画素の各サブ画素位置に対して別々のカラー値及びアルファ値のアキュムレータを有し、各サブ画素について別々にカラー値を累積する。このサブシステムは、各サブ画素位置の累積されたカラー値を組み合わせるロジックを備え、最終的な出力画素を計算する。別の手法として、フラグメント解析サブシステムは、深度値でソートしたリストにおいて各フラグメント記録を解析する際に、共通の累積したアルファ値を持つサブ画素領域を追跡し続ける。リストの中の各フラグメントを解析した後、二つの手法で得られた出力は、ただ一組のカラー値(ＲＧＢ)、及び、可能であれば、１つのアルファ値を有する出力画素となる。各画素位置に対し、例えば、ＲＧＢカラー値とアルファ値を含む解析した画素値を計算するために、フラグメント解析サブシステムは、画素バッファ内のカラー値を、関連するフラグメントリスト内のフラグメント記録におけるいずれかのフラグメントと組み合わせる。

【００２７】

本発明のさらに別の態様は、異方性フィルタを実行する方法にある。一般にテクスチャ・マッピングにおいて、グラフィックス・レンダリングシステムは、幾何形状プリミティブの面に対して、テクスチャ・マップをマップする。この特定の方法においては、本システムはまず、ビュー空間内の画素位置の一点をどのようにテクスチャ・マップにマップさせるかを決定する。概念的には、本システムは、フィルタ・フットプリントをどのように

テクスチャ・マップにマップさせるかを決定する。透過マッピングにおいては、等方性フィルタ・フットプリントをテクスチャ・マップにマップさせることは、異方性の方向においては歪んだ形を与える。したがって、テクスチャを等方性フィルタでフィルタすることは、高品質の結果を得るには十分ではない。ある特定の実施例では、本システムは、フィルタ・フットプリントをテクスチャにマップさせる手法について、テクスチャ座標にマップされたビュー空間座標（例えば、画面座標）における画素位置に対する逆ヤコビアン行列を計算することによって決定する。

【００２８】

次に、本システムは、マップされたフィルタ・フットプリントから異方性直線を決め、特に、この特定の一実施例においては、逆ヤコビアン行列によって異方性直線を決める。概念的には、異方性直線は、ビュー空間からテクスチャ空間へとマップされた点の座標を通過する直線であり、マップされたフィルタ・フットプリントの最大伸長方向に方向付けされる直線である。本システムは、異方性直線に沿ってフィルタリングを、テクスチャ・マップからのサンプル数値に対し、繰り返し適用する。この繰り返しフィルタリング・ステップによる出力は、最終のテクスチャ値を計算するために、フィルタリングされ、累積される。この手法に関しては、多くの変形がある。ある特定の実施例では、本システムは、異方性直線に沿ってトライ・リニア補間を実行する。次に、このトライ・リニアフィルタの出力は、画素位置に対する一組のカラー値を計算するように組み合わせられる。この態様においては、テクスチャフィルタ・エンジンは、例えば、三角形又は台形の形を持つ一次元フィルタを、異方性直線に沿ってトライ・リニア補間の出力に適用する。しかしながら、異方性直線に沿って適用されるフィルタに関しては、この方法を用いて多くの変形が可能である。

【００２９】

本発明の別の態様は、ある画像が、フラグメントメモリからオーバフローした場合に、本システムは、その画像の小部分のレンダリングを実行する手法にある。一実施例では、本システムは、フラグメントメモリの使用状況を追跡し、フラグメント・エントリの数がある所定値に達した場合、画像領域を、小部分にサブ分割することができる。本システムは、画素フラグメントを生成しながら、フラグメントバッファへのエントリの数を追跡し続ける。エントリの数が所定値に達したならば、画像領域はさらに小領域にサブ分割され、このサブ分割した小領域について1回に１つずつレンダリングする。これによって、各サブ分割した小領域をレンダリングするのに十分なフラグメントメモリが確保される。本システムは、フラグメント・エントリの数が所定値に達したならば、小領域をさらに小さい画像領域にサブ分割することができる。従って、本システムは、フラグメントメモリの容量を越えることがないことを確実とするように、レンダリングされる画像領域をサブ分割することができる。これによって、上記の工夫を持たなかったならば、フラグメントメモリがオーバフローしてしまうような場合でも、本システムであれば、フラグメントを捨て去ることなく、より小さいフラグメントメモリを用いることができる。

【００３０】

本発明の別の態様は、本システムは、高いレイテンシを有する環境において、テクスチャ・フェッチ動作を実行する手法にある。例えば、テクスチャ・マッピング、シャドウイング、マルチパス・レンダリングなどの動作においては、それら動作を実行するためにテクスチャデータをフェッチする際に、高いレイテンシを有する場合がしばしばある。このレイテンシは、メモリからデータを読み取る際に生ずる遅延、テクスチャデータを復元する際に生ずる遅延、又は、その双方によって生じるものである。

【００３１】

一実施例においては、入力データ・ストリームの中の幾何形状プリミティブは、メモリから１ブロックのテクスチャデータをフェッチする際のレイテンシを十分吸収できる長さのプリミティブキューに蓄えられる。プリラスタライザが、このプリミティブキュー内の幾何形状プリミティブを、テクスチャブロック参照データにレンダリングし、これが、第２のキューに蓄えられる。この第２のキューで参照データとされたテクスチャブロックは

、メモリからフェッチされ、テクスチャキャッシュに置かれる。ポストラスタライザは、このキュー内のプリミティブを１つずつラスタライズする。各プリミティブがラスタライズされるにつれて、現在のプリミティブ用の出力画素の計算のために、テクスチャデータが必要に応じてテクスチャキャッシュから取り出される。プリミティブは、ラスタライズされた後、このキューから除去される。

【００３２】

別の一実施例では、プリミティブがラスタライズされることにより、得られた画素データは、テクスチャブロック・フェッチ・ユニットのレイテンシを吸収するのに十分な長さのキューに置かれる。ある特定の態様では、キューへのエントリは、画素アドレス、そのアドレスに対するカラーデータ、及び、テクスチャ要求を含む。テクスチャ要求はさらに、テクスチャ・マップ座標におけるテクスチャサンプルの中心点を含む。テクスチャ要求は、テクスチャブロック・アドレスに変換され、このテクスチャブロックは、フェッチされ、テクスチャキャッシュに置かれる。キューにおけるエントリは、キューから取り出され、この時点でテクスチャキャッシュにある関連するテクスチャデータは、出力画素を計算するのに用いられる。双方の実施例とも２組のテクスチャ要求を生成し、その内の一方の組は、他方の組のために遅延させられる。第１の組は、テクスチャデータを実際にフェッチし、可能であれば、それを復元するのに用いられ、 第２の組は、テクスチャキャッシュからテクスチャデータを取り出すのに用いられる。

【００３３】

本発明の、更なる特徴及び利点に関しては、下記の詳細な説明と図面を参照することによって明らかになる。

【実施例】

【００３４】

装置概観

下記の詳細な説明の中で、画像処理システムに関して、幾つかの実施例を説明する。

この画像処理システムは、グラフィックスとビデオ処理の双方に対して、リアルタイム画像のレンダリングと生成をサポートする。本システムに用いられる新規のアーキテクチャと画像処理法によって、本システムは、従来のグラフィックスシステムに比べて、高度なリアルタイム３Ｄアニメーションを、はるかに低いコストで生成することができる。画像処理に加えて、本システムは、ビデオ編集アプリケーションのようなビデオ処理をサポートし、また、ビデオとグラッフィクスを組み合わせることもできる。例えば、本システムは、ビデオをグラッフィクオブジェクトに適用するように、又は、ビデオデータにグラッフィクオブジェクトを加えるように用いることもできる。

【００３５】

本システムは、広範なインタラクティブ・アプリケーションをサポートする。本システムは、高度のリアルタイム・アニメーションをサポートできるために、ゲームや、教育用アプリケーション、及び、多くのインタラクティブ・アプリケーションに好適である。本システムは、３Ｄグラッフィクス、及び、グラフィックスとビデオの組み合わせを含む、高度なユーザ・インターフェイスをサポートする。従来の、パソコン用ウィンドウ環境の制限されたグラフィックス能力を改良し、本システムは、改良された、アプリケーション用３Ｄグラフィック・ユーザ・インターフェイスをサポートする。そのアプリケーションは、デスクトップ・コンピュータ上のオフィス・インフォメーション処理用のものから、セット・トップ・ボックスにおけるインタラクティブ・テレビジョン用アプリケーションのものまで広範に渡る。本システムは、メモリと処理時間を極めて効率よく利用するので、アプリケーションの性能や、ユーザ・アクションに対するユーザ・インターフェイスの反応性を不当に妨げることなく、際立った画像処理能力及び表示能力をもたらす。

【００３６】

図１は、本画像処理システム１００ のブロック図である。本画像処理システムは、画像データ供給・保存装置１０２、画像前処理装置１０４、画像処理装置１０６、及び、レンダリングされた画像を直ちに表示したい場合の表示装置１０８からなる。本システムの

各構成要素は、システム・インターフェイス１１０ を通じて互いに接続されている。画像データ供給・保存装置１０２は、画像データを本システムに供給し、かつ、画像データ及び各種命令を保存する。画像前処理装置１０４は、画像データを操作して、レンダリングできるように調整する。前処理機能の例としては、オブジェクトを幾何形状モデルで規定すること、ライティング及びシャドウイング形成モデルを規定すること、オブジェクト位置を決めること、視点と光源の位置を決めること、幾何学処理、がある。

画像処理装置１０６は、画像をレンダリングし、表示装置１０８に表示されるべき表示画像を生成する。レンダリングとは、モデルから画像を生成する処理を指し、幾つか挙げるならば、幾何学処理（幾何学処理は、前処理機能でもあることに注意）、可視面の決定、走査変換、及び、ライティングなどの機能を含む。画像、又は、画像の一部をレンダリングした後、画像処理装置１０６は、レンダリングした画像データを、表示用の表示装置に転送する。

【００３７】

以下に、画像処理システム１００の幾つかの特質を、特定のハードウェア・ソフトウェア・アーキテクチャを参照しながら詳細に説明する。しかしながら、下記の画像処理システムは、各種のアーキテクチャに実装可能であることを理解しておくことが重要である。

【００３８】

画像処理システム１００は、本発明者にとって既知の従来の高品質３Ｄグラフィックスシステムに対して、はるかに優るコストパフォーマンス上の改善を達成した。コンピュータ・グラッフィクスにおける幾つかの進歩がこの改善の達成に貢献している。そのような進歩としては、合成画像層、画像圧縮、チャンキング、及び、マルチパス・レンダリングがある。ここに、これらの進歩を紹介し、以下にさらに詳細に、これらの進歩、並びに、他の進歩を説明する。

【００３９】

合成画像層（ｇスプライト）

本システムにおいては、多数の、独立の画像層がビデオ速度で合成されて、出力ビデオ信号を生成する。この画像層を、一般化したスプライト(generalized sprite)、即ちｇスプライト(gsprite) と呼ぶことにするが、これらは、独立にレンダリング及び操作することができる。本システムは、包括的には、場面における各非貫通性のオブジェクトに対して、個々のｇスプライトを用いる。これによって、各オブジェクトを、独立に更新することができるので、オブジェクトの更新速度を、場面の優先性に応じて最適化することができる。例えば、はるか遠くの背景において動きあるオブジェクトについては、前景オブジェクトほどには、頻繁に、又は、それほど正確に更新する必要はない。

【００４０】

ｇスプライトは任意のサイズ及び形状とすることができる。ある実現例では、方形のｇスプライトを用いた。ｇスプライトの画素は、その画素に関連するカラー及びアルファ（不透明度）情報を持っており、そのために、多数のｇスプライトが合成されて全体場面を構成できるようになっている。

ｇスプライトに対しては、拡大・縮小、回転、サブ画素位置づけ、アフィン・ワープのような小さな動きに対する変換を含む、幾つかの異なる動作をビデオ速度で実行することができる。従って、ｇスプライトの更新速度を変える一方で、ｇスプライトの変換（動きなど）は、完全なビデオ速度で実現可能なので、更新速度の保証を持たない従来の３Ｄグラフィックスシステムで達成できるレベルをはるかに越えた流動力学を実現する。

【００４１】

多くの３Ｄ変換は、２Ｄ画像処理演算によってシミュレートすることができる。例えば、後退するオブジェクトは、ｇスプライトの大きさを拡大・縮小させることによってシミュレートすることができる。中間フレームに対しては、前にレンダリングした画像に対して２Ｄレンダリングを実行することによって、全体の処理要求を大きく低減させることができるので、３Ｄレンダリング性能を、最高品質の画質を生成することが要求される場合に用いることができる。これは、ディテール管理の時間レベルの態様である。

【００４２】
ｇスプライトの拡大・縮小を用いることによって、ディテールの空間レベルも、場面の優先性に適合するように調整することができる。例えば、背景オブジェクト（曇った空など）は、小さなｇスプライト（低解像度）へとレンダリングでき、次に、そのｇスプライトを、表示用に適当な大きさに拡大・縮小する。高品質フィルタリング機能を用いることによって、典型的な低解像度アーティファクトも目立たなくなる。
通常の、３Ｄグラッフィクス・アプリケーション（特にインタラクティブ・ゲーム）は、高度のアニメーション速度を達成するために、ディテールの幾何学的レベルをトレードオフにしている。ｇスプライトによって、本システムは、付加的に、２つの場面パラメータを利用することができる。即ち、ディテールの時間レベルと、ディテールの空間レベルである。そして、それらによって、ユーザから見たときの性能を効果的に最適なものとする。あるオブジェクトの画像がレンダリングされる時の空間解像度は、そのレンダリングの際に画面解像度に適合させる必要はない。さらに、本システムは、アプリケーション側のサポートを必要とせず、自動的にこれらトレードオフを管理することができる。
【００４３】
画像圧縮
グラフィックスシステムのコストパフォーマンスの決定において、最上位の要素は恐らくメモリであろう。例えば、従来の高性能３Ｄグラフィックスシステムは、フレームバッファ（ダブルバッファ）、深度バッファ、テクスチャバッファ、及び、アンチエイリアシングバッファを含めて、３０メガバイトを越えるメモリを持つ。そして、この多くは、専用メモリであって、ＤＲＡＭよりもずっと高価である。メモリバンド幅は常に致命的なボトルネックとなる。高性能グラフィックスシステムのコストに関して、画素及びテクスチャデータ・アクセスのために十分なバンド幅をもたらすように、インターリーブ・メモリの多数のバンクを設ける必要性に駆り立てられることが多い。
【００４４】
本システムは、この問題を解決するために、画像圧縮技術を広く用いている。画像圧縮は、従来、グラフィックスシステムには使用されなかった。これは、高品質を実現するのに要求される計算上の複雑さのためであり、また、圧縮技術そのものが、従来のグラフィックス・アーキテクチャに容易に適合しないためでもあった。そこで、チャンキング（後述）と呼ぶ概念を用いることによって、画像やテクスチャに対して圧縮を効果的に適用することができ、それによって、コストパフォーマンス面において著しい改善を実現することができた。
【００４５】
一態様においては、グラフィックスシステムは従来から、フレームバッファ・メモリに対して圧縮を用いてきた。高性能グラフィックスシステムは、三つのカラー成分のそれぞれに対して８ビットを用い、さらに、８ビットのアルファ値を含めることも多い。低性能グラフィックスシステムは、情報を捨てることによって、及び／又は、カラー・パレットを用いることによって、画素当たり３２ビットを４ ビットまで圧縮し、同時に表示可能なカラーの数を少なくすることによって実行する。この圧縮は、極めて目立つアーティファクトを生じさせ、データ要求に対して十分な抑制ができず、それでいて、アプリケーション及び／又はドライバに、広範な画素フォーマットに対処するように強制する。
【００４６】
本システムで使用される圧縮は、高画質を実現しながら、１０：１ 以上の圧縮比を与える。本システムの別の利点は、全てのアプリケーションに対して、単一の高品質の画像フォーマットしか使用しないということである。これは、空間解像度とカラー値・深度値との間のトレードオフを必要としていた標準のPCグラフィックス・アーキテクチャとは、明確に異なる点である。
【００４７】
チャンキング
本システムの、別の重要な進歩を、チャンキングと呼ぶ。従来の３Ｄグラフィックスシ

ステム（その点に限って言えばどのフレームバッファでも）ランダムにアクセスが可能である（されている）。画面上の任意の画素に対して、ランダムな順序でアクセスすることができる。圧縮アルゴリズムは、相当に大きな数の隣接画素へのアクセス性に依存するので（空間コヒーレンスを利用するために）、圧縮は、グラフィックス・アルゴリズムによって用いられるランダム・アクセス・パターンの性質のために、全ての画素の更新がなされて始めて実行可能となる。このため、圧縮技術を、表示バッファに対して適用することは実用的でなくなる。

【００４８】

また、このようなランダム・アクセス・パターンは、画素単位の隠れ面消去や、アンチエイリアシング・アルゴリズムを導入した場合、画面上の各画素に対して、余分の情報を維持しなければならなくなることを意味している。これは、メモリ・サイズ要求を飛躍的に増大させ、さらに性能上のボトルネックを加えることになる。

本システムは、別の方法を取る。ある場面、又は、場面の一部は、チャンクと呼ばれる画素領域に分割される（ある特定の実現例では３２×３２画素）。一実現例では、システムは、ｇスプライトに割り当てられた幾何形状を複数のチャンクに分割するが、別の実現例では、ｇスプライトを用いずにチャンキングを実行する。この幾何形状は、どのチャンクにレンダリングされるかに基づいてビンに予めソートする。この処理をチャンキングと呼ぶ。チャンク境界をオーバーラップする幾何形状は、オーバーラップが見られる各チャンクを参照することが好ましい。場面が活発に変化するにつれて、本システムによれば、データ構造を、１つのチャンクから別のチャンクへと移動する幾何形状に対して調整するように修正する。

【００４９】

チャンキングによって、幾つかの著しい利点が得られる。チャンキングの使用によって、効果的な態様の圧縮を提供することができる。１つのチャンク内の幾何形状は全て、後続するチャンクに進む前にレンダリングされるため、深度バッファは、僅か、単一チャンクだけの大きさを持つだけでよい。３２×３２画素のような比較的小さなチャンク・サイズを用いることによって、深度バッファを、グラフィックス・レンダリング・チップに直接実装させることができる。これによって、かなりの量のメモリを削減することができ、さらに、深度バッファを、専用メモリ・アーキテクチャを用いて実現することもできる。即ち、そのメモリ・アーキテクチャは、きわめて高いバンド幅をもってアクセスすることができ、かつ、ダブルバッファ動作時には空にすることができるので、フレーム間のオーバーヘッドを無くすことで、従来のフレームバッファ・メモリを削減することができる。

【００５０】

アンチエイリアシングも相当に簡単になる。なぜなら、各チャンクは、独立に処理できるからである。アンチエイリアシング機能を備える、ほとんどの高性能Ｚ バッファグラフィックスシステムは、大量の付加メモリを利用しながら、比較的単純なフィルタリングしか実行できない。しかしながら、チャンキングを用いれば、必要とされるデータ量は、かなり少ない（１０００の桁）ので、さらに高度なアンチエイリアシング・アルゴリズムの実装を可能にする。

【００５１】

Ｚバッファ処理やアンチエイリアシングに加えてさらに、本システムは、適正な、シームレスの態様で半透明処理をサポートする。あるチャンクを形成中、本システムは、もう１つのチャンクでアンチエイリアシングと半透明処理の計算を行なう。換言すれば、ある１つのチャンクを形成するのに必要な時間内で、本システムは、もう１つのチャンクでアンチエイリアシングと半透明処理を実行することができる。本システムは、チャンク間をピンポンのように往復することができるので、リアルタイム・アプリケーション用に画像を処理するにあたって、遅延を加えることなく高度の処理を実行することができる。

【００５２】

さらに別の利点は、チャンキングは、ブロック性の画像圧縮を可能にすることである。一旦チャンクがレンダリングされた（かつ、アンチエイリアシングされた）ならば、次に

そのチャンクは、ブロック変換ベースの圧縮アルゴリズムによって圧縮される。したがって、チャンクを別々にチャンキングすることによって実行される圧縮に加えて、チャンキングがさらに高度な、適応性の高い圧縮方式をサポートすることになる。

【００５３】

マルチパス・レンダリング

本システムのアーキテクチャのもう１つの利点は、３Ｄインタラクティブ・アプリケーションが、１９７０年代終わりに見られるＣＡＤグラフィックスシステムの画像表現―あの、飽き飽きする、フォン・ハイライト（Phong highlight）を備えた、ランバーティアン・グーローシェーディングによるポリゴン群（lambertian Gouraud-shaded polygons）―から抜け出すためのきっかけを与えることである。カラー値のテクスチャ・マッピングは、この画像表現を改善するばかりでなく、アプリケーション上ではさらにもう１つ独特の画像表現を呈する。１９８０年代に、プログラム可能なシェーダ(shader)や、処理用テクスチャ・マップのアイデアが、レンダリング処理に新たな飛躍をもたらした。これらのアイデアは、オフライン・レンダリングの世界を一掃し、今日、映画の特殊効果に見るような高品質画像を生成させる。

【００５４】

今日における通常の高性能３Ｄ画像ワークステーションの、厳密なレンダリング・パイプラインモード、及び、固定のレンダリングモードは、このような効果を実現した場合、必ずリアルタイム性能を急激に低下させる結果を招く。そのため、リアルタイム表示を要求するユーザは、狭く制限されたレンダリングの融通の利く範囲内に甘んずるを得ない。

【００５５】

上に概説した方法を用いてバンド幅要求を下げることによって、本発明のシステムは、単一の共用メモリを、圧縮テクスチャの保存、及び、圧縮ｇスプライトの保存を含む、全てのメモリ要求を満たすように用いることができる。このアーキテクチャは、レンダリング処理で生成されたデータが、テクスチャ処理装置に戻され、新たなｇスプライトのレンダリングのためのデータとして使用されることになる。このようなフィードバック・サポートのために、本システムは効率的なマルチパス・レンダリングを実行することができる。

【００５６】

効率的なマルチパス・レンダリングを、様々の合成モードと柔軟性のあるシェーディング言語とを組み合わせることによって、本システムは、以前はオフ・ライン・ソフトウェア・レンダリングの領域であった、各種のレンダリング効果をリアルタイムで供給することができる。そのようなものとして、シャドウ（複数光源によるシャドウイングを含む）、環境マップの反映オブジェクト、スポット・ライト、地上の霞、リアルな水中シミュレーションなどの機能のサポートがある。

【００５７】

一実施例においては、画像処理システム（１００）は、ソフトウェアとハードウェアの組み合わせを含む。次節で、ハードウェアとソフトウェア・アーキテクチャを参照しながら、システム環境について述べる。可能な限り、別態様のアーキテクチャについても述べる。しかしながら、ソフトウェアとハードウェアは変更が可能であるから、後述する特定の実施例には限定されない。

本画像処理システム、又は、その部分は、デスクトップ・コンピュータ、セット・トップ・ボックス、及び、ゲームシステムを含む、幾つかの異なるプラットフォームに実装することができる。

【００５８】

図２は、本画像処理システムが実装されるホスト・コンピュータシステム１３０のブロック図である。ホスト・コンピュータシステム１３０は、プロセッサ１３２、メイン・メモリ１３４、メモリ制御装置１３６、二次保存装置１３８、入力装置（単数又は複数）、表示装置１４２、及び、画像処理ハードウェア１４４を含む。メモリ制御装置１３６は、プロセッサ１３２とメイン・メモリ１３４との間のインターフェイスとなる。メモリ制御

装置はさらに、プロセッサ１３２とメイン・メモリ１３４が、バス１４６に接する時のインターフェイスとしても働く。

【００５９】

図２に示したと同一、又は、同様のアーキテクチャを持つコンピュータシステムは様々ある。そのようなシステム内部のプロセッサも変動し得る。さらに、１個以上のプロセッサを含むコンピュータシステムもある。幾つか挙げるならば、プロセッサ１３２は、インテル社のペンティアム（登録商標）、又は、ペンチティアム・プロであってもよいし、シリコン・グラフィックス社のＭＩＰＳ系列から成る小型プロセッサであってもよいし、モトローラ社のパワーＰＣであってもよい。

【００６０】

メイン・メモリ１３４は、高速メモリであって、ほとんどの通常のコンピュータシステムでは、ランダム・アクセス・メモリ(RAM) が装備されている。メイン・メモリは、既知の技術のどのような変種においても、プロセッサ及びバスとインターフェイスを形成する。メイン・メモリ１３４は、コンピュータのオペレーティング・システムや、現に実行するアプリケーション・プログラムのようなプログラムを保存する。以下に、コンピュータシステムによって実行される命令のシンボル表現を参照しながら、一実施例の様々な態様を述べる。この命令は、コンピュータ実行とも呼ばれる場合がある。この実施例のこれらの態様は、コンピュータ読み取り可能な媒体に保存された一連の命令からなる、１つのプログラム又は複数のプログラムとして実現することもできる。コンピュータ読み取り可能な媒体は、メイン・メモリ又は二次保存装置と組み合わされる、ここに説明する装置等のいずれであっても、又は、それら装置等の組み合わせであってもよい。

【００６１】

バス１４６は、メモリ制御装置１３６、二次保存装置１３８、及び、画像処理ハードウェア１４４を互いに結び付ける。ある実現例では、例えば、バスはＰＣＩバスである。ＰＣＩ標準は既知であり、この標準をサポートするように設計されたコンピュータシステムのボードはいくつかある。例としては、ＩＳＡバス、ＥＩＳＡバス、ＶＥＳＡローカル・バス、及び、ＮｕＢｕｓがある。

表示装置１４２は、カラー表示装置であって、画像を表示するために連続的に更新される。一実施例の表示装置は、陰極線管(ＣＲＴ)を用いた表示装置であるが、これは、液晶表示(ＬＣＤ) であっても、その他の表示装置であってもよい。

二次保存装置１３８は、各種の保存媒体を含む。例えば、二次保存装置は、フロッピー（登録商標）・ディスク、ハードディスク、テープ、ＣＤ－ＲＯＭなどを含んでいてよく、また、電気的、磁気的、光学的、又は、他の記録材料を用いる装置であってもよい。

【００６２】

入力装置(単数又は複数)１４０は、キーボード、マウスや、ジョイスティックのようなカーソルポジショニング装置の他に、各種市販の入力装置を含む。

下記に詳述する１つの実現例においては、画像処理ハードウェア１４４は、コンピュータシステムと、ＰＣＩバスを通して結合するボードに実装される。

また別の実現例では、画像処理ハードウェアは、プロセッサや、他の画像処理ハードウェアやメモリと共にシステム・ボードに置かれる。例えば、あるゲームシステムでは、画像処理ハードウェアは、通常、マザー・ボード上に置かれる。同様に、セット・トップ・ボックスの画像処理ハードウェアもマザー・ボード上に置いてもよい。

【００６３】

コンピュータシステムのアーキテクチャを概説したからといって、本発明を、図２に示したシステム・アーキテクチャに限定するつもりはない。本画像処理システムは、ゲームシステム、セット・トップ・ボックス、ビデオ編集装置などに実装することができる。以下に、図２に示したシステム・アーキテクチャの環境における画像処理システムの実施例を説明する。下記の説明全体を通じて、別態様の実施例を述べるが、他に考えられる可能な実現例の完全なリストを与えるつもりで、これら別態様の実施例を説明するものではない。下記の詳細な説明に基づけば、当業者であれば、本画像処理システムを、別のプラッ

トフォームに実現することは可能である。

【００６４】

図３は、一実施例におけるソフトウェアとハードウェアの間の関係を示すブロック図である。この実施例においては、画像処理システムは、ホスト・コンピュータのプロセッサと、画像処理ハードウェア１４４の処理リソースに基づいて実現される。画像処理ハードウェア１４４は、プロセッサ(例えば、ディジタル信号プロセッサ１６６)と画像処理回路１６８を含む拡張ボード上に実装される。ホスト・コンピュータシステム１３０と、画像処理ボードのプロセッサは、画像処理演算を共同で行なう。以下に、ホスト・コンピュータシステム１３０と画像処理ボード１７４によって実行される機能を概説する。

【００６５】

グラッフィクス・サポート・ソフトウェア１６０は、ホスト・コンピュータシステム１３０の上で実行され、ハードウェア抽象化層(ＨＡＬ)１６２を通じて画像処理ボード１６４と情報を交換する。画像処理ボード１６４は、ＤＳＰと呼ばれるプログラム可能なディジタル信号プロセッサ１６６と、以下に詳述する別の画像処理ハードウェア１６８を含む。

【００６６】

グラフィックス・サポート・ソフトウェア１６０は、メモリ管理、視体積の選択、深度のソーティング、チャンキングの他に、ｇスプライトの割り当て、変換、及び、ディテールのレベルをサポートする機能を含む。グラフィックス・サポート・ソフトウェアは、ここに挙げる機能を実行するために、グラフィックス・アプリケーション類のアクセス可能な、一群のグラフィックス機能を含んでいてもよい。

【００６７】

グラフィックス・サポート・ソフトウェア１６０は、上に紹介したｇスプライト・パラダイムをサポートする機能を含む。上に示したように、ｇスプライトは独立にレンダリングされるが、フレームごとにレンダリングする必要はない。むしろ、ｇスプライトの位置の変化は、アフィン変換、又は、その他の変換で近似させてもよい。グラフィックス・サポート・ソフトウェア１６０は、１つのオブジェクト、又は、複数のオブジェクトを、１つのｇスプライトに割り当てることを助ける機能と、そのｇスプライトの位置と動きを記述する移動データを追跡する機能とを備える。さらに、グラフィックス・サポート・ソフトウェアは、レンダリングされたｇスプライトを更新すべき時を決める機能を備える。ｇスプライトを更新する必要のある場合は、オブジェクトの動き、視点の動き、ライティングの変化、及び、オブジェクトの衝突に応じて変化する。

以下に、グラフィック・サポート・ソフトウェアの機能に関してさらに詳細を述べる。画像処理ボード１６４は、変換、ライティング、及び、シェーディング、テクスチャリング、アンチエイリアシング、半透明処理などを含む低レベルの幾何学処理を実行する。一実施例では、ＤＳＰ１６６が、フロントエンド幾何学処理とライティング計算を実行するが、この機能のうちいくつかは、プロセッサ１３２に実行させてもよい。

【００６８】

画像処理ボードの概観

図４Ａは、画像処理ボード１７４を示すブロック図である。この画像処理ボード１７４は、ホスト・コンピュータと、バス１４６を介して連絡する。この画像処理ボード１７４は、ＤＳＰ１７６、タイラー２００、共用メモリ２１６、ｇスプライト・エンジン２０４、合成バッファ２１０、及び、ディジタル・アナログ変換器(ＤＡＣ)２１２を含む。ホストからの命令に応じて、画像処理ボード１７４は、画像をレンダリングし、表示用画像を、ＤＡＣ２１２を介して表示装置１４２(図２)に転送する。

【００６９】

図２～４Ａに示す実施例では、プロセッサ１３２とＤＳＰ１６６は、図１の画像前処理装置１０４の機能を共有している。画像処理装置１０６は、タイラー２００、ｇスプライト・エンジン２０４、合成バッファ２１０及びＤＡＣ２１２から成る。以下に、これらの構成要素についてさらに詳細を述べる。しかしながら、画像処理システムの実現は変更可

能であることに留意すべきである。

【００７０】

共用メモリ２１６は、画像処理ボード１７４上において、画像データ及び画像処理命令を蓄える。一実施例では、共用メモリは、ｇスプライトとテクスチャデータを圧縮した形で蓄えるように用いられ、各種バッファは、処理サブシステム間にデータを転送するように用いられる。

ＤＳＰ１７６は、ビデオ圧縮・復元と、フロントエンド画像処理（変換、ライティング、など）を行なう。できれば、ＤＳＰは、１０００ ＭＦＬＯＰＳ／ＭＯＰＳを越える浮動小数点と積分計算をサポートするものであることが好ましい。

タイラー２００は、ＶＬＳＩチップであって、マルチパス・レンダリングのために、走査変換、陰影階調、テクスチャリング、隠れ面の消去、アンチエイリアシング、半透明処理、シェーディング、及び、マルチパス・レンダリングを行なう。次に、この結果生じるレンダリングしたｇスプライト・チャンクを圧縮し、圧縮した形で共用メモリに蓄える。タイラーはさらに、ビデオ、及び、ウィンドウ演算をサポートするために、ｇスプライト・データの復元及び再圧縮を行なう。

【００７１】

ｇスプライト・エンジン２０４は、ビデオ速度で、ｇスプライトのチャンク・データに接触し、これを圧縮し、一般のアフィン変換（これには、拡大・縮小、画素以下の精度を持つ変換、回転、反射及びせん断（shearing）が含まれる）のために必要な画像処理を行なう。フィルタリング後に得られた画素（アルファ値を含む）は、合成バッファに送られ、ここで表示画素データが求められる。

【００７２】

ｇスプライト・チャンク・データは、表示のために、１回に走査線の数だけ処理される。一実施例では、チャンク・データは、１回に３２本の走査線分処理される。合成バッファ（２１０）は、二種の、３２走査線カラーバッファを含み、これら二種を、表示と合成動作の二つの間で交互に切り替える。合成バッファはさらに３２走査線アルファ・バッファを含む。このバッファは、各画素に対するアルファ値を累積するように用いられる。

ＤＡＣ２１２は、ＲＧＢ ビデオＤＡＣと対応するビデオ・ポート２１４から、ビデオ編集装置までを含む。個々の成分は、ＤＡＣの機能性を実現するために用いられる。

【００７３】

システムの動作

図５Ａ及び図５Ｂは、本画像処理システムにおいて画像をレンダリングする際に実行されるステップを示すフローチャートである。画像処理装置１０６が、ビュー空間に対して画像をレンダリングする前に、画像前処理装置１０４は、オブジェクトと視点の位置を決める(２４０)。図２及び図３で示した実施例では、ホスト・コンピュータシステム１３０ で実行するグラフィックス・サポート・ソフトウェア１６０が、グラッフィクス・アプリケーションから供給されるデータに基づいて、オブジェクトと視点の位置を決める。プロセッサ１３２で実行するグラフィックス・アプリケーションは、関係するオブジェクトを表わすモデルを規定し、モデル変換を供給して、そのオブジェクトを、他のオブジェクトと共に「ワールド座標」の中に位置づけるように使用される。

【００７４】

次に、画像前処理装置１０４は、潜在的に可視のオブジェクトを選び出す(２４２)。画像前処理装置１０４は、視体積に基づいて、潜在的に可視のオブジェクトを決める。視体積とは、ワールド座標における三次元空間であって、場面に対して境界を与える。画像前処理装置１０４は、オブジェクトをトラバースすることによって、潜在的に可視のオブジェクトを選び出し、その境界が、視体積と交差するかどうかを決める。視体積と交差するオブジェクトは、幾何学的又は空間的な意味で、潜在的に可視であることなる。

【００７５】

ある場合には、場面の将来の変化を予測するために、現在の視体積の外部に、「時間的に」、潜在的に可視のオブジェクトを決めておくことが有利なことがある。こうすれば、

本システムは、視体積の急激な変化にも対応していくことができる。通常の３Ｄグラフィックスシステムでは、このような急激な変化に対応する唯一の手法は、変更された入力に基づいて新たな場面を完全に生成させることであり、そのため、その間に相当な伝達遅延を生ずることになる。そのような長い遅延は、ユーザに否定的効果を及ぼし、調整過度や気分のむかつきのような問題を生じさせる。この遅延を低減させるために、本発明の画像前処理装置１０４は、可視域の外側へと拡大した範囲に置かれるオブジェクトの位置を計算することができ、かつ、画像処理装置１０６は、この拡大範囲の中の画像をレンダリング及び保存することができる。本システムのアフィン変換機能に基づき、次のフレームに対する視点の入力を、この伝達遅延を低減させる拡大範囲からのｇスプライトを、計算フレームにして２フレーム未満に再配置するように用いることができる。そのような短い伝達遅延は、本発明者にとって既知の従来の３Ｄグラッフィクス・ハードウェアシステムでは到達不可能なものであり、さらに高品質なシミュレーションを可能にし、したがって、ユーザをさらに心地よく画面に没頭させることができる。

【００７６】

画像前処理装置１０４は、画像に対するｇスプライトの構成を決定する(２４４)。このステップは、潜在的に可視のオブジェクトをｇスプライトにマップする方法を見つけ出すステップを含む。この処理の一部として、画像前処理装置１０４は、ｇスプライトの割り当てを行なって、ｇスプライトのデータ構造を生成し、それによって1 つ以上の潜在的に可視のオブジェクトに対応する画像データを蓄えることを含む。処理用リソースが許すならば、場面内の非貫通性オブジェクトの各々は、独立のｇスプライトに割り当てられる。非貫通性、又は、自己遮蔽オブジェクトは、単一のｇスプライトとして処理される。

【００７７】

画像前処理装置１０４は、画像処理装置１０６が、これらｇスプライトを所望の計算フレームレートで合成する能力を持たない場合、あるいは、それらｇスプライトを蓄えるほど十分なシステムメモリがない場合、これらｇスプライトを合体させることができる。別々のｇスプライトにレンダリングすることは、必ず計算上効率が高くなるので、システムにメモリ能力や、合成能力があるならば、非交差性オブジェクトを、別々のｇスプライトにレンダリングする。システムが、ｇスプライトの現在の割り当てに基づいて表示画像を保存ないし生成できない場合には、幾つかのｇスプライトを合体して、この問題を緩和する。

【００７８】

１つのオブジェクト、又は、複数のオブジェクトをｇスプライトに割り当てた後で、画像処理装置１０６は、ｇスプライトを、「チャンク」と呼ばれる画像領域に分割する（２４８）。画像前処理装置１０４が、ｇスプライトに対しループ（繰り返し演算）を実行し、このｇスプライトをチャンクに分割する（２４６、２４８）。一実施例では、この処理は、ビュー空間に対するオブジェクトの境界ボリュームを変換し、この変換された境界ボリュームを包む方形の画像領域を見つけ出すことを含む。この画像領域は、ｇスプライトのオブジェクト（単数又は複数）がレンダリングされる二次元空間におけるｇスプライトの大きさを定める。ｇスプライトは、この方形画像領域をチャンクに分割することにより、それらチャンクをｇスプライト・データ構造と関連付けて、チャンクに分割される。

【００７９】

最適化の１つの手法として、変換された境界ボリュームを拡大・縮小し、及び／又は、回転させ、それによって、ｇスプライト変換に必要なチャンクの数が極小になるようにしてもよい。このように変換（拡大・縮小、あるいは、回転）をさらに加えることから、ｇスプライトに割り当てられたオブジェクトの空間は、必ずしも画面空間である必要はない。この空間を、ｇスプライト空間と云う。表示画像を生成する処理で、ｇスプライトを再び変換して画面空間に戻さなければならない。

【００８０】

次のステップは、オブジェクトの大きさを、チャンク間にどのように分割するかを決めることである（２５０）。画像処理装置１０６は、幾何形状プリミティブ（例えば、ポリ

ゴン）が、チャンク間にどのように分割されるかを、そのポリゴンを２Ｄ空間（２５２）に変換し、そのポリゴンがどのチャンク（単数又は複数）に投影するかを求めることにより、決定する。ポリゴンを切り取るのは高コストであるから、好ましい方法は、チャンクの端にあるポリゴンは切り取らないことである。むしろ、チャンクは、その端を重ねる複数のポリゴンを含む。例えば、１つのポリゴンが、２つのチャンクの境界を超えて延在している場合は、この方法では、ポリゴンの頂点が、各チャンクに含まれることになる。

【００８１】

次に、画像前処理装置１０４は、チャンク・データを、タイリングのためにキューする。タイリングとは、１つ以上のポリゴンで全体に、又は、部分的にカバーされる画素位置の、カラー値やアルファ値のような画素値を決める処理を云う。

決定ステップ（２５４）（図５Ｂ）、及び、それに続くステップ(２５６）は、チャンク内でポリゴンのタイリングを行なう処理を表わす。画像処理装置１０６は、現在のチャンクの境界に重なるポリゴンを含む一方、同チャンク内にある画素しか生成しない。生成された画素は、アンチエイリアシング(フラグメント記録) のための情報を含み、フラグメント記録は、全ての画素が生成されるまで保存される。

【００８２】

チャンクのポリゴンのタイリングが終了した後、画像処理装置１０６は、画素のためのアンチエイリアシングデータ（フラグメント記録など）を解析する(２５８)。一実施例では、タイラー２００ は、ダブルバッファを用い、次のタイリングを行なっている間に、その前のチャンクを解析する。また別のやり方として、タイラーは、フリー・リストを有する共用バッファを用いることができる。フリー・リストとは、共有バッファ内の空のメモリを表わし、これは、新しいフラグメント記録が生成されると割り当てられ、フラグメント記録が解析されると、また加えられるものである。ダブルバッファ法と共有メモリの組み合わせを用いても同様の効果が期待できる。

【００８３】

画像処理装置１０６は、後述する圧縮法を用いて、この解析されたチャンクを圧縮する。画像処理装置１０６は、１ブロックの画素を解析すると同時に、別のブロックを圧縮してもよい。画像処理装置１０６は、共用メモリにこの圧縮チャンクを蓄える(２６２）。

図６は、画像を表示するために実行されるステップを示すフローチャートである。上述の画像処理ボード１７４上において、画像は、共用メモリ２１６から読み取られ、ｇスプライト・エンジン２０４によって、物理的出力装置座標に変換され、合成バッファ２１０において合成され、ＤＡＣ２１２に転送され、次にその出力装置に転送される。

【００８４】

表示処理において、画像処理装置１０６は、現在のフレームに表示されるべき一連のｇスプライトにアクセスする。ｇスプライト形態を決定する処理において、画像前処理装置１０４は、ｇスプライトの深度の桁を決める（２８０）。前述したように、１つのｇスプライトには１つのオブジェクトを割り当てるのが好ましい。しかしながら、画像前処理装置１０４は、例えば、本システムで使用される、特定の画像処理装置の処理限界を受け容れるために、１つのｇスプライトに対して１個以上のオブジェクトを割り当ててもよい。画像前処理装置は、オブジェクトを、Ｚ順に、即ち、視点からの距離として、オブジェクトをソートする。オブジェクトをソートするのに加えて、さらに、ｇスプライトを深度順にソートし、この深度データをｇスプライト・データ構造に蓄える。

【００８５】

図６の決定ステップ(２８２) は、表示処理におけるｇスプライトに対するループを表わす。このループ内のステップとしては、１)レンダリングしたｇスプライト用の変換を求めること、２)如何にｇスプライトを表示するかを制御するためのｇスプライト表示リストを形成すること、がある。これらのステップについては後述する。

潜在的に可視範囲のｇスプライトに関しては、画像処理装置１０６は、ｇスプライト変換を求める。ｇスプライト変換とは、レンダリングされた２Ｄのｇスプライトにおける変換を云う。一実施例では、画像処理装置１０６は、レンダリング・オーバーヘッドを低減

するようにｇスプライトに対して変換を実行する。フレームごとに各オブジェクトをレンダリングするのでなく、画像処理装置１０６は、レンダリングされたｇスプライトを再利用することによって、レンダリング・オーバーヘッドを低減させる。

【００８６】

画像データの各フレームにｇスプライト変換を計算する必要はない。例えば、ｇスプライトが、画像データの現在フレームに対してレンダリングされたのであれば、そのｇスプライトを変換する必要はない。これは、例えば、そのｇスプライトをレンダリングして、境界ボックスをそのオブジェクトによりよく適合させようという要求がない限りそうである。さらに、再レンダリングや変換を要しないｇスプライトがある。即ち、そのｇスプライトに割り当てられたオブジェクト（単数又は複数）が変化しない場合や、動かない場合である。そのような場合、ｇスプライトを変換するステップは、実行してもよいし、しなくともよい。

ｇスプライトの位置が変化しない場合には、このｇスプライトに、単位行列をかけてもよい。これは、例えば、画像処理装置１０６が、現在のフレームのためにｇスプライトをレンダリングした場合、あるいは、始めにレンダリングされて以来、ｇスプライトの位置が変化しない場合に適用される。

【００８７】

ｇスプライトの表示法を特定するために、画像処理装置１０６は、ｇスプライト表示リストを生成する。表示リストとは、どのｇスプライトを、表示画面に表示すべきかを規定する１つのリスト、又は、複数のリストを云う。この表示リストの概念は、1フレームの画像データを表示するための他の出力装置にも適用することができる。画像処理装置１０６は、表示リストを、レンダリングされたｇスプライトを物理的出力装置座標にマップし、さらに合成するのに用いる。表示を構成させるステップは、ｇスプライトに対するループの一部として示されているが、このリスト（単数又は複数）は、特にこのループ内に生成される必要はない。

【００８８】

表示リストは、一連のｇスプライト・リスト、又は、バンド幅単位の、一連のｇスプライト・リストであってもよい。「バンド幅」とは、表示画面の水平走査線領域である。例えば、一実施例では、バンド幅は、３２走査線高 ×１３４４画素幅である。表示リストは、各バンド幅で別々のｇスプライト・リストを含んでいてもよく、その場合、バンド幅リストは、それぞれのバンド幅に照射するｇスプライトを記述する。また別法として、表示・リストは、どのバンド幅にｇスプライトが照射するのかを特定できるようにｇスプライトに名札をつけて実行される、そのような単一リストからなるものであってもよい。

【００８９】

ここに示した実施例の表示リストは、ダブルバッファ方式である。ダブルバッファを持つことによって、本システムは、他方のデータ・リストを読み取っている間に、一方のデータ・リストを生成することができる。本システムが１つのフレームに対するｇスプライトの変換を求め、表示リストを構成している間に、別のフレームの表示リストを読み取り、このリストの中の画像データを表示する。

ダブルバッファであるために、図６に示したステップは重複する。即ち、画像処理装置１０６は、あるフレームに対しては、ステップ(２８０－２８６）を実行する一方で、別のフレームに対してはステップ(２９０－２９８)を実行する。

【００９０】

図７は、上記ステップのタイミングを示すブロック図である。本システムが、フレーム３１０ に対してステップ(２８０－２８６)(図６) を完了すると、本システムは、フレーム同期信号(垂直点線) を待ち、それからバッファ交換を実行する。次に、本システムが生成したばかりの表示リストを用いて、現在フレーム３１２ に表示すべきｇスプライトを決める。この表示リストを処理している間に３１２、ｇスプライト変換が計算され、かつ、次のフレームのための表示リストが構成される３１４。次のフレームにおいて、前のフレーム３１４で生成されたｇスプライト変換と表示リストが用いられ、表示画像３１６

を生成する。

【００９１】

画像処理装置１０６は、表示リストの中のｇスプライトのリストに基づいて、ｇスプライトを出力装置座標に変換する。画像処理装置１０６は、カラー値、アルファ値、及び、ｇスプライトの位置を特定するデータを含むｇスプライト・データを共用メモリから読み取る。このデータに基づいて、画像処理装置１０６は、そのｇスプライトによってカバーされる画素のカラーとアルファを決める。

一実施例においては、画像処理装置１０６は、各バンド幅にループ制御を行ない、ｇスプライト表示リストに従って、そのバンド幅に照射するｇスプライトを変換する。以下に、この表示処理については、さらに詳細に述べる。

【００９２】

ｇスプライト・データを変換した後、画像処理装置１０６は、得られた画素データを合成する。この中には、ｇスプライト変換に基づいて、出力装置座標で表わした画素に対するカラー値とアルファ値を計算することを含まれる。画像処理装置１０６は、表示リストの中のｇスプライトに対する画素データを変換し、次に、その変換した画素データを合成する。この処理は、ある画素位置について、それをカバーするｇスプライトの１つ以上の画素値の組み合わせに基づいて、その位置のカラー値とアルファ値を決定することを含む。

【００９３】

一実施例では、画像処理装置１０６は、バンド幅にループ制御を実行し、各バンド幅に画素データを合成する。画像処理装置１０６は、画素データをダブルバッファリングで扱う。即ち、１つのバッファにおいてあるバンド幅のｇスプライトを変換及び合成する一方で、別のバンド幅に対しては、合成済み画素データを表示する。

画素データを合成した後、画像処理装置１０６は、合成画素データを物理的出力装置に転送する。画素データを表示するためには、それは、表示と適合するフォーマットに変換されていなければならない。

【００９４】

一実施例のシステムの全体動作を説明し終えたので、次に、画像処理ボードについてさらに詳細に述べる。

画像処理ボード

一実施例では、共用メモリ２１６は、４メガバイトのＲＡＭを含む。これは、２本の８ビットＲＡＭバス・チャンネルによって実行される。しかしながら、メモリの量及び型は変更可能である。

【００９５】

図８は、画像処理ボード１７４におけるＤＳＰ３３６を示すブロック図である。ＤＳＰ３３６は、プロセッサ１３２からの命令ストリームを取りまとめ、若干のビデオ処理とフロントエンド幾何学処理を実行する。ＤＳＰは、３Ｄグラフィックスに用いられるフロントエンド幾何学処理、及び、ライティング計算を実行する。この中には、モデル変換及びビューイング変換や、クリッピング、及び、ライティングが含まれる。ｇスプライトの動き外挿のような、ｇスプライト・アニメーション処理の部分もＤＳＰで処理される。

【００９６】

レンダリング命令は、メインメモリ・バッファに保存され、ＰＣＩバスを通り、ＰＣＩバス制御装置３４２を経由して画像処理ボード１７４にＤＭＡ化される。次に、これらの命令は、ボードの共用メモリ２１６に、ＤＳＰ３３６(図８) に要求されるまで、蓄えられる。

ＤＳＰコア３３８は、上述の画像処理計算を実行するためのプロセッサを含む。さらにＤＳＰコアは、スケジュール設定、及び、リソース管理を実行する。

【００９７】

メモリ・インターフェイス３４０は、高速のデータ転送、例えば、８０ＭＨｚにおいて６４ビットをサポートする。これは、通常のＤＲＡＭやＳＤＲＡＭ装置とインターフェイ

スを形成するように設計されている。タイラー２００は、このバスに直接接続するように設計されており、ＤＳＰによって要求されるメモリ・タイミングをシミュレートする。

【００９８】

ＤＳＰにおけるデータ・フォーマッタ及びコンバータ３４６は、タイラーに対するレンダリング命令をフォーマットする。このブロックは、浮動小数点カラー成分を、整数に変換し、それらを、タイラー専用データ構造にパッキングする。さらに、全命令を蓄え、これを直接共用メモリ中のメモリ・バッファにＤＭＡする。このレンダリング命令は後に、動作実行の態勢が整った時に、タイラーによって読み取られる。

【００９９】

フォーマットするタスクの間で、データ・フォーマッタとコンバータ３４６は、タイラーに対して三角形・命令データをフォーマットする。ＤＳＰ(３３６)によって浮動小数点として計算されたＲＧＢ ∝（アルファ値）データは、８ビットの整数に変換される。座標情報は、浮動小数点から１２．４の固定点に変換される。このデータは、６４ビット語にパックされ、一連のブロックとして、共用メモリに転送され、これにより、最適なバンド幅が得られる。

【０１００】

表示メモリ管理部（ＭＭＵ）３３４は、デスクトップ表示メモリのために使用される。表示メモリ管理部（ＭＭＵ）３３４は、デスクトップ表示メモリとして割り当てられた、ラインアドレス範囲内のＰＣＩアクセスをトラップする。次に、このアクセスを、共用メモリに蓄えられた画像ブロックにマップする。

画像処理ボード１７４（図４Ａ）のアーキテクチャは、特定のＤＳＰからは比較的独立している。しかしながら、ＤＳＰは、顕著な浮動小数点性能を持っていることが好ましい。適当なＤＳＰとしては、サムスン社（Samsung）製 の半導体のＭＳＰ－１、及び、フィリップス社（Phillips）製の半導体のトリメディア（Trimedia）が挙げられる。これら特定のＤＳＰは、十分な浮動小数点性能を与えるＤＳＰの二つの例である。

【０１０１】

図９Ａは、画像処理ボード１７４におけるタイラー200 のブロック図である。タイラーは、２Ｄ及び３Ｄグラッフィクスの加速と、共用メモリ調整を実行する。画像処理ボード１７４のブロック図で示したように、タイラーは直接ＤＳＰ１７６（図４）、ｇスプライト・エンジン２０４、及び、共用メモリ２１６に接続する。

【０１０２】

上記ブロック図の機能ブロックは、この節に記載される。

タイラー３７８は、プリミティブ・レンダリングのために多くの構成要素を含む。命令・メモリ制御装置３８０は、共用メモリ２１６、ｇスプライト・エンジン２０４、及び、ＤＳＰ１７６とインターフェイスを形成する。タイラー、ＤＳＰ、及び、ｇスプライト・エンジンからの、メモリに対するアクセスは、このブロックによって判断される。キューが、バッファ読み取りアクセスに対して与えられる。

【０１０３】

セットアップブロック３８２は、三角形の面を横切るエッジ、カラー、及び、テクスチャ座標の補間を決める直線方程式を求める。この方程式はまた、この三角形をレンダリングするにはどのテクスチャブロックが必要とされるかを決めるのにも用いられる。エッジ方程式はさらに走査変換ブロック３９４にも送られ、走査変換エンジン３９８に要求されるまで、プリミティブレジスタ３９６に蓄えられる。

【０１０４】

セットアップブロック３８２は、三つの構成要素を含む。頂点入力処理装置３８４、頂点制御レジスタ３８６、及び、セットアップ・エンジンである。頂点入力処理装置３８４は、ＤＳＰからの命令ストリームを集める。頂点制御レジスタ３８６は、ポリゴン、又は、その他の幾何形状プリミティブを処理するのに必要な情報を蓄える。この特定の実施例では三角形の処理が用いられ、かつ、タイラー２００は、三角形の処理のダブルバッファ対応を実行するために、６頂点（各三角形につき三つずつ）のレジスタを含む。セットア

ップ・エンジン３８８は、カラー、深度、エッジに対する微分と、三角系の面を横切るテクスチャ座標の補間を計算する。これらの方程式はまた、この三角形をレンダリングするにはどのテクスチャブロックが使ったらよいかを決めるのにも用いられる。セットアップ・エンジンはさらにテクスチャチャンクをあらかじめフェッチし、それによって、走査変換エンジン３９８に要求された場合にいつでも利用可能なようにしておく。

【０１０５】

セットアップ・エンジン３８８はさらにテクスチャ読み取りキュー３９０、及び、テクスチャアドレス生成装置３９２と連絡する。テクスチャ読み取りキュー３９０は、共用メモリのテクスチャブロック読み取り要求を保存する。ここで、メモリから画像データブロックを取り出すために用いられるタイラーの部分を指して「テクスチャ」という用語を使用する時は、この用語は、マルチパス・レンダリング動作で用いられるテクスチャ・マップ、シャドウマップ、及び、その他の画像データを云うこともあり得るということを了解しなければならない。テクスチャアドレス生成装置392 は、要求されたチャンクの、メモリ内部のアドレスを決め、テクスチャ読み取り要求を、命令・メモリ制御装置３８０に送る。このテクスチャアドレス生成装置３９２は、画像データの、テクスチャキャッシュへの書き込みを制御するメモリ管理部を含む。

【０１０６】

走査変換ブロック３９４は、セットアップブロックから微分値とその他の頂点データを受け取り、画素データを生成する。走査変換ブロック３９４は、プリミティブレジスタ396 と、走査変換エンジン３９８とを含む。プリミティブレジスタ３９６は、各三角形のパラメータに対する方程式パラメータを蓄える。プリミティブレジスタは、複数組の方程式を蓄えるためのレジスタを含み、それによって、走査変換エンジンが、テクスチャデータを待ち続けて停止することがないようにする。

【０１０７】

走査変換エンジン３９８は、ポリゴンを変換する。この場合、ポリゴンは三角形である。走査変換ブロック３９４は、移動エッジ用、又は、カラー、深度などを評価用の補間器を含む。カラー、深度を備えた画素アドレス、及び、アンチエイリアシング包括情報は、画素エンジンに転送され、処理される。

走査変換エンジン３９８は、テクスチャアドレスを、テクスチャデータを算出するテクスチャフィルタ・エンジン４００に送出する。テクスチャフィルタ・エンジン４００は、レンダリングしたポリゴンの画素のカラー値及びアルファ値を求める。図に示したテクスチャフィルタ・エンジンは、レンダリングした三角形のＺ勾配と方向、及び、テクスチャ要求の中心に基づいてフィルタ・カーネルを計算する（１点のＳ とＴ座標がテクスチャにマップされる）。フィルタリングは、パイプライン方式によって、各サイクルに新しい画素が生成されるように二段階で実施される。フィルタ・カーネルは、異方性フィルタであってもよいし、等方性フィルタであってもよい。異方性が要求されない場合は、フィルタ・カーネルは、トライ・リニア補間で可能とされる場合よりも更に鋭いテクスチャを可能とするネガティブローブ（negative lobes）フィルタを用いてもよい。テクスチャフィルタ・エンジン４００はさらに、シャドウに対する効果を計算するようにＺ比較動作を処理する。

【０１０８】

テクスチャキャッシュ４０２は、復元画像データのブロックを蓄える。一実現例では、テクスチャキャッシュ４０２は、１６個の、８×８画素・ブロックのテクスチャデータを蓄える。このデータは、１６個のテクスチャ要素が、クロック・サイクルごとにアクセスされるようにまとめられる。

【０１０９】

復元エンジン４０４は、テクスチャデータを復元し、これを、テクスチャキャッシュ４０２に転送する。本実施例においては、復元エンジンは、２つの復元器を含む。一方は、テクスチャのような連続階調画像に対して、離散コサイン変換（ＤＣＴ）準拠アルゴリズムを実行するもので、他方は、デスクトップ・画素データ用に脱落のないアルゴリズムを

実行する。ＤＣＴ準拠アルゴリズムは、二つの、並行な復元ブロックによって実行され、しかも、このブロックは各々クロック・サイクルごとに８画素要素（即ち、２画素）を生成できる。

【０１１０】

圧縮キャッシュ４１６は、復元エンジン４０４が圧縮データを復元し、テクスチャキャッシュ４０２に転送するまでは、その圧縮データを蓄えるのに使用される。

走査変換エンジン３９８は、画素データを画素エンジン４０６に転送する。画素エンジン４０６は、混合、及び、深度のバッファリングを含む画素レベルの計算を実行する。画素エンジンはさらにシャドウに必要なＺ比較動作を処理する。最適性能を発揮するためには、画素エンジンは、クロック・サイクルごとに１画素ずつ動作することが好ましい。

【０１１１】

画素エンジン４０６は、ラスタライゼーションバッファへの画素データの転送を制御する。ここに示した実施例では、ラスタライゼーションバッファは、画素バッファ４０８とフラグメントバッファ４１０を含む。画素バッファ４０８は、ダブルバッファリングをサポートするために２個のバッファを含む。この画素バッファを実現例において、各画素エントリは、カラー成分（ＲＧＢ）当たり８ビット、アルファ成分用に８ビット、Ｚ バッファ用に２４ビット、ステンシル・バッファ用に８ビット、フラグメントバッファ用に９ビット・ポインタを蓄える。これは、画素当たり合計７３ビットになる。一方の画素バッファが画素エンジン４０６で使用されている時、他方は、アンチエイリアシング・エンジン４１２によって使用される。

【０１１２】

フラグメントバッファ４１０は、画素フラグメントと呼ばれる、部分的にカバーされる画素のフラグメントを蓄える。このようなフラグメントは、ポリゴンのエッジが、ある画素を跨る場合、又は、ポリゴンが、半透明性を持つ画素である場合である。フラグメントバッファは、図９Ａに示した実現例では単一バッファ処理で実行される。フラグメントのフリー・リストの保持は次のように行なわれる。即ち、フラグメントが解析されるにつれて、それらはフリー・リストに加えられ、フラグメントが生成されるにつれて、そのフラグメントは、フリー・リストからのエントリを利用する。また、別法として、フラグメントバッファはダブルバッファリング方式であってもよい。それによって、他方のフラグメントバッファが、並行的に画素エンジンによって処理がいっぱいとされた際に、一方のフラグメントバッファをアンチエイリアシング・エンジンによって解析することができる。

【０１１３】

一実施例では、フラグメント記録は、画素バッファ・エントリに、４×４マスクを加えたものと同じデータを含む。９ビット・ポインタは、リストの終端を示すリザーブ値を有する、リンク化したエントリのリストを形成するように用いられる。この実施例では、フラグメントバッファ４１０は、合計５１２個のエントリを含むが、その大きさは変動してもよい。

【０１１４】

アンチエイリアシング・エンジン４１２は、１個以上のポリゴンによって悪影響を被る画素におけるカラーとアルファ成分を求める。このようなことは、複数のポリゴンが、その画素領域をほんの部分的にしかカバーしない時（即ち、ポリゴンのエッジがその画素を横切る時）、又は、ポリゴンが半透明性を持つときに起こる。アンチエイリアシング・エンジン４１２は、解析された画素データを圧縮エンジン４１４に転送する。この実施例では、圧縮エンジン４１４は、二つの圧縮器を含む。即ち、一方は、連続階調の画像用のＤＣＴ準拠のものであり、他方は、デスクトップ・画素データ用のロスレス・タイプのものである。ＤＣＴ準拠アルゴリズムは、クロック・サイクル当たり８画素要素を圧縮できる圧縮器によって実行される。圧縮エンジン４１４は、結果として生じるレンダリングしたｇスプライトを圧縮し、この圧縮データを命令・メモリ制御装置３８０に転送し、これは、共用メモリ２１６に保存される（図４）。タイラーも、圧縮データをキャッシュに換える圧縮キャッシュ４１６を持つ。

【０１１５】
図１０及び図１１は、画素生成処理において、メモリから画像データにアクセスするに当たって、二つの別態様の実施例を示したものである。画素生成中にメモリから画像データにアクセスしなければならない場合がいくつかある。そのような例としては、例えば、テクスチャ・マッピング動作中にテクスチャ・マップにアクセスすること、シャドウイング動作中にシャドウマップにアクセスすること、及び、マルチパス・レンダリング動作中に、カラー値、及び／又は、アルファ値にアクセスすることが挙げられる。簡単のために、メモリ中の画像データを、「テクスチャ」又は「テクスチャデータ」と呼ぶ。しかしながら、ここに述べる方法ならびにシステムは、画素生成中に、メモリから画像データにアクセスする他のタイプにも応用できることを了解しなければならない。

【０１１６】
図１０及び図１１に示した実現例は、タイラーに対しテクスチャキャッシュを効率よくロードし、利用するための別法を紹介するものである。この方法の大きな利点は、テクスチャデータを高いレイテンシを有するメモリに保存でき、本システムの性能を損なうことなく、圧縮形式でさえ保存することができる。従って、専用とすることなく、低コストのメモリを、高性能のレンダリング用ハードウェアに実装するように用いることができる。

【０１１７】
メモリからのテクスチャデータは、「ブロック」と呼ばれる単位ごとにアクセスされ、キャッシュされる。このブロックは、通常、効率的なフェッチや、キャッシュに好適な小さな方形領域である。通常のブロックの大きさは、８×８サンプルである。例えば、テクスチャ・マップの場合では、通常のブロックは、８×８テクセルである。

【０１１８】
図１０は、一実施例において、このようなテクスチャデータのブロックにアクセスする態様を示す機能的ブロック図である。この実施例は、テクスチャデータ要求を含むラスタライザ４１７からの画素データを、テクスチャ参照データ・キュー４１８に蓄えることによってレイテンシ問題を解決している。このキューは、テクスチャ参照データ・キュー４１８がなければテクスチャブロックにアクセスする際に（また、可能であれば、復元する際に）、生じるレイテンシを吸収するに十分なエントリを含んでいるために、レンダリング処理はフル・スピードで実行することができる。例えば、１つのテクスチャブロックをフェッチするのに１００サイクルを要し、タイラーが、クロック・サイクル当たり１画素を生成できるとするならば、テクスチャ参照データ・キュー４１８は、少なくとも１００個のエントリを含むことになる。

【０１１９】
図１０に示した装置においてデータの流れは、下記のように進行する。先ず、幾何形状プリミティブが、ブロック４１６に示すようにラスタライズのためにセットアップされる。セットアップ処理としては、例えば、三角形のような幾何形状プリミティブに対して頂点を読み取ること、及び、その三角形の面を横切ってカラー、深度とエッジの微分を求めること、が挙げられる。この計算によって得られるパラメータを次にラスタライザ４１７に入力する。

【０１２０】
ラスタライザ４１７は、各プリミティブに対して方程式パラメータ・データを読み取り、画素データを生成する。ラスタライザは、テクスチャ座標とフィルタ・データを含む画素データを生成し、このデータを、テクスチャ参照データ・キュー４１８に蓄える。テクスチャ・フェッチ・ブロック４２０は、テクスチャ参照データ・キュー４１８に蓄えられるテクスチャ参照データを読み取り、かつ、メモリ４１９から適当なテクスチャブロックをフェッチする。

【０１２１】
本実現例のテクスチャ参照データ・キュー４１８に蓄えられる画素データは下記を含む。即ち、計算された、画素の宛先アドレス（Ｘ，Ｙ）、深度データ（Ｚ）、カバレッジマスク、カラー値と半透明性データ、テクスチャ要求の中心座標（Ｓ，Ｔ）、及び、テクス

チャフィルタ・データである。深度データ、カバレッジマスクのデータは、画素の高度なアンチエイリアシングを望む場合にのみ、テクスチャ参照データ・キューにおいて要求される。隠れ面消去処理やアンチエイリアシングをラスタライザ内で実行する場合、深度データや、カバレッジマスクのデータは、テクスチャ参照データ・キュー４１８内に保存する必要はない。テクスチャフィルタ・データは、例えば、ＭＩＰマッピング用の一定レベルのディテールのパラメータを含んでいてもよいし、或いは、さらに高度なテクスチャフィルタのための異方性フィルタ・データを含んでいてもよい。

【０１２２】

テクスチャブロック・フェッチ・ユニット４２０は、データ・キューに蓄えられるテクスチャ参照データを読み取り、メモリ４１９から対応するテクスチャデータを取り出す。テクスチャ・マップ・アクセスの場合、テクスチャブロック・フェッチ・ユニットは、テクスチャ要求の中心（Ｓ，Ｔ）とテクスチャフィルタ・データを、テクスチャフィルタ動作を満足に実行するのに必要なブロックのアドレスに変換する。画像データブロックは、最近使用したキャッシュ(ＬＲＵ)、又は、その他の適当なキャッシュ置換アルゴリズムを用いてフェッチすることができる。メモリアクセスの回数を抑えるために、テクスチャブロック・フェッチ・ユニットは、テクスチャキャッシュ４２１にすでに蓄えられているテクスチャブロックを追跡し続けており、同じブロックを１回よりも多く要求するのを避ける。この能力によって、高品質のテクスチャフィルタを実行するのに必要なバンド幅を大幅に下げることができる。なぜなら、テクスチャブロックを取り出す際のレイテンシが、画像を計算する時の１回だけしか起こらないからである。

【０１２３】

テクスチャブロック・フェッチ・ユニットは、タイラーのテクスチャフィルタ・ユニットでまだ必要とされているテクスチャブロックの書き込みすぎを防止するために、抑制機構を備える。このような抑制機構を実行する1つの手法は、各テクスチャブロックと参照カウントを関連させ、テクスチャフィルタ・ユニットが、特定のテクスチャブロックを用いたかどうかを追跡し続けることである。この参照カウントは、ブロックに対するテクスチャ要求を、テクスチャブロック・フェッチ・ユニットが受け取る度ごとに増加し、そのブロックがテクスチャフィルタ・ユニットによって使用される度ごとに減少する。従って、テクスチャブロック・フェッチ・ユニットは、対応する参照カウントがゼロのブロック同士を交換すればよい。

【０１２４】

抑制機構を実行するまた別の方法として、テクスチャブロック・フェッチ・ユニットからのテクスチャブロック出力のために、テンポラリバッファを割り当てることがある。この方法では、画像ブロックは、最初に、テンポラリバッファに書き込まれる。テクスチャブロック・フェッチ・ユニットが、このテンポラリバッファへの画像ブロックの書き込みを完了した後で、この画像ブロックはテクスチャキャッシュに転送される。画像ブロックは、テクスチャフィルタ・ユニット４２２に最初に要求された時に、テクスチャキャッシュと交換される。

【０１２５】

テクスチャ・マッピング動作の場合、テクスチャフィルタ・ブロック４２２は、テクスチャキャッシュ４２１からテクスチャサンプルと、さらに、テクスチャ参照データ・キュー４１８ に蓄えられている画素データを読み取り、そのテクスチャサンプル・データから画素カラー値及び、可能であれば、アルファ値を計算する。

テクスチャ・マッピング動作の他にも、この方法は、シャドウイングや、マルチパス・レンダリング動作にも同様に適用できる。例えば、テクスチャ参照データ・キューを用いて、メモリの中にあるシャドウの深度マップを取り出すことができる。また別法として、テクスチャ参照データ・キューは、マルチパス動作のライティング及びシャドウイングで使用されたカラー値、及び／又は、アルファ値を取り出すように用いることができる。テクスチャ・マッピング、シャドウイング、及び、マルチパス動作についてはさらに詳細に以下に述べる。

【０１２６】
前記の方法で画素データを保存すると幾つかの利点が得られる。１つの大きな利点は、画像データを、あまり特別でない（比較的大きなアクセス時間のものでもよい）メモリに保存することができることである。これは、システム全体のコストを下げる。さらに、テクスチャを含む画像データを、圧縮フォーマットで保存でき、従って、テクスチャフィルタのような高度な画素の処理を実行できるほどに高速でアクセスすることができることである。このため、本システムは、テクスチャデータにアクセスするに当たって、既知の方法に比べて、より低いコストでより向上した性能を発揮することができる。
【０１２７】
この方法の別の利点は、テクスチャ参照データ・キューが、メモリからどの画像ブロックにアクセスしなければならないかを正確に予測することができることである。そのため、本システムは、メモリアクセスのために必要以上のレイテンシを生じさせない。一旦画像データブロックがテクスチャキャッシュの中に入ったならば、要求される画像ブロックをテクスチャキャッシュに書き込むのに十分なメモリバンド幅とテクスチャ・フェッチのスループットがある限り、テクスチャフィルタ・ユニットは、ラスタライザをフル・スピードで実行することができる。
【０１２８】
テクスチャ参照を、テクスチャ要求の中心でキューすること、及び、テクスチャ参照データをフィルタリングすることは、テクスチャ参照データ・キューが、同じテクスチャフィルタ重量のままでテクセルをキューした場合よりも、極めて小さくすることができる。
【０１２９】
図１１は、もう一実施例において、メモリから画像データにアクセスするステップを示す機能的ブロック図である。この方法では、幾何形状プリミティブは、キューされ、プリラスタライザ４２６において、画素生成処理で生じたテクスチャブロック・フェッチ・ユニットのレイテンシをマスクするように処理される。一例を挙げて説明すると、この概念が具体的に明らかになるであろう。 平均的なプリミティブが、ラスタライズするのに２５サイクルを要し、かつ、メモリからテクスチャブロックをフェッチするのに１００クロック・サイクルを要するとするならば、このプリミティブキューは、少なくとも４つ分のプリミティブの長さとなる。ポストラスタライザ４２７の単純型としてのプリラスタライザは、メモリからアクセスするのに必要な画像データブロックを決める回路を含む。一旦テクスチャデータがフェッチされたならば、ポストラスタライザは、メモリからブロックをフェッチする際に生ずるレイテンシをも生ずることなく、テクスチャデータを用いて画素データを生成することができる。
【０１３０】
この実現例全体を通ずるデータの流れは下記のようになる。前記の実現例同様、幾何形状プリミティブは、セットアップブロック４２５で処理され、ラスタライズされる。しかしながら、この特定の実現例では、セットアップブロック４２５は、前例よりもより多くのプリミティブを蓄えることのできる大きなプリミティブキューを含む。プリラスタライザ４２６は速やかにこれらプリミティブを、一連のテクスチャブロック・リストに変換し、一連のテクスチャブロック・リストは、プリミティブによってカバーされる全ての画素に対して、それらブロックがポストラスタライザ４２７によって必要とされる順に、テクスチャフィルタリング要求を満たすのに必要とされるものである。プリラスタライザ４２６は、ポストラスタライザ４２７の単純型、又は、前述した別の実施例のラスタライザ４１７の単純型である。この方法では、プリラスタライザは、テクスチャデータ・アドレスを計算し、かつ、テクスチャ要求を決めるだけでよい。
【０１３１】
プリラスタライザもテクスチャブロック・キャッシュのモデルを保持し、キャッシュ置換アルゴリズム、例えば、最近使用したキャッシュ(ＬＲＵ) 方式を実行し、テクスチャブロック・キャッシュの大きさを越えないようにする。キャッシュ置換アルゴリズムの一部として、プリラスタライザは、テクスチャブロック・フェッチ・ユニットへの単一のテ

クスチャブロックに対する繰り返し要求を、ただ１つの要求に圧縮する。

【０１３２】

テクスチャブロック・フェッチ・キュー４２８は、テクスチャブロック要求を蓄えるエントリを備える。テクスチャブロック・フェッチ・ユニット４２９は、テクスチャブロック・フェッチ・キューからのテクスチャ要求を読み取り、メモリ４３０から適当なブロックを取り出す。

【０１３３】

ポストラスタライザ４２７は、セットアップブロック４２５でキューされたプリミティブをラスタライズし、ある画素位置に対する画素データを生成する。画像データを、画素生成処理中に、メモリからアクセスする必要があれば、ポストラスタライザは、必要なテクスチャブロックがテクスチャブロック・キャッシュ４３１に転送されるのと同一の速さで、このプリミティブをラスタライズする。ポストラスタライザが、セットアップブロックでキューされたプリミティブをラスタライズし終えると、プリミティブは除去され、入力データ・ストリームからの別のプリミティブと交換される。プリラスタライザ及びポストラスタライザが、画素生成処理において待ち続けて停止することないように、このセットアップブロックは、プリミティブで満たされるキューを保持する責任を有する。

【０１３４】

前述した別態様の実施例の場合と同様に、テクスチャブロック・フェッチ・ユニット４２９は、ポストラスタライザ４２７によってまだ必要とされているテクスチャブロックの超過することを防ぐための抑制機構を含むことが好ましい。前述の２個の抑制機構は、この実施例でも使用できる。特に、画像ブロックがいつ要求され、そして、いつ使用されるかを追跡し続けるように、参照カウントを用いることができる。その場合、参照カウントは、ブロックに対するテクスチャ要求を、プリラスタライザ４２６が受け取る度ごとに増加し、ポストラスタライザ４２７が使用する度ごとに減少する。従って、テクスチャブロック・フェッチ・ユニット４２９は、テクスチャキャッシュのブロック同士を、その参照カウントが０の場合にのみ交換すればよい。

【０１３５】

また、別法として、バッファを、テクスチャ・フェッチ・ブロック４２０によるテクスチャブロック出力の一時的保存のために割り当ててもよい。テクスチャ・フェッチ・ブロック４２０が、このテンポラリバッファへのテクスチャブロックの書き込みを完了すると、そのテクスチャブロックは、ポストラスタライザ４２７によって要求される場合、テクスチャブロック・キャッシュ４３１に転送される。ポストラスタライザ４２７が最初にテンポラリバッファ中のテクスチャブロックを要求すると、そのテクスチャブロックは、テクスチャブロック・キャッシュ４３１に転送される。

【０１３６】

この方法にはいくつか利点がある。先ず、テクスチャデータを、あまり専用化されていないメモリに保存しても、高度なテクスチャフィルタをサポートするのに十分な速度でアクセスすることができる。それに関連する重要な利点として、テクスチャデータは、圧縮フォーマットで保存することができ、かつ、画素生成処理で使用の際に復元されることである。

【０１３７】

この方法のもう１つの利点は、メモリに対する要求が予測できるために、メモリアクセスによるレイテンシを、場面をレンダリングする各テクスチャブロックに対して、ただ一度だけしか生じさせないで済むということである。一旦、最初のテクスチャブロックがテクスチャキャッシュの中に格納されたならば、ポストラスタライザ４２７は、メモリバンド幅を有し、キャッシュの流れを保持するためのテクスチャ・フェッチスループットを有する限り、フル・スピードで実行することができる。

【０１３８】

図９Ｂは、図１０に示した本システムの実現例をさらに詳細に図示したものである。図９Ｂのセットアップブロック３８１は、図１０のセットアップブロック４１６に相当する

。図９Ａのセットアップブロック３８２と違って、この別種実現例のセットアップブロック３８１は、テクスチャ読み取り要求を生成しない。代わりに、走査変換ブロック３９５が、テクスチャ参照データを含む画素データを生成する。このデータは、テクスチャ参照データ・キュー３９９に蓄えられる。

【０１３９】

図９Ｂの走査変換ブロック３９５は、図１０のラスタライザのある特異な実現例である。走査変換ブロック３９５は、Ｚ値、カバレッジマスク、カラー・半透明性データ、及び、テクスチャ座標で表わされたテクスチャ要求の中心を計算する。幾つかのテクスチャ・マッピング動作では、このブロックは、さらにディテールのレベルのデータや、異方性フィルタ・データも計算する。テクスチャフィルタ・エンジン４０１は、テクスチャ要求と、通常、テクスチャ参照データ・キュー３９９に保存されているテクスチャフィルタ・データを読み取り、テクスチャキャッシュの中の適当なテクスチャサンプルにアクセスする。このテクスチャデータから、テクスチャフィルタ・エンジンは、テクスチャがどの程度、画素カラー値及びアルファ値に関与するかを計算する。テクスチャフィルタ・エンジンは、テクスチャ参照データ・キュー３９９の中のカラー値及びアルファ値を、テクスチャからのカラー値及びアルファ値と組み合わせて、画素値を生成し、その画素値を画素エンジン４０６に送出する。

テクスチャキャッシュ制御装置３９１、テクスチャ読み取りキュー３９３、及び、命令・メモリ制御装置３８０は、図１０のテクスチャブロック・フェッチ・ユニット４２０の特異な実現例である。さらに、圧縮テクスチャブロックに対しては、圧縮キャッシュ４１６と復元エンジン４０４が、テクスチャブロック・フェッチ・ユニット４２０の一部となる。

図９Ｃは、図１１に示した本システムのさらに詳細な実現例を示す。この実現例では、図１１のブロック４２５及びブロック４２６に関連して説明する機能は、セットアップブロック３８３内部に実装される。特に、セットアップブロック３８３は、プリラスタライザ４２６を含む。セットアップブロック３８３はさらに、追加のプリミティブを含む付加的な頂点制御レジスタ３８７を含む。そのために、プリラスタライザは、プリミティブを速やかに変換し、テクスチャデータ要求を発することができる。セットアップ・エンジンとプリラスタライザ３８３は、テクスチャブロックに対する要求を、図９Ｃに示したテクスチャキャッシュ制御装置３９１に送出する。

【０１４０】

テクスチャキャッシュ制御装置３９１は、要求されたテクスチャブロックが、必要な時には、テクスチャキャッシュ４０２に入っていることを確かめる。テクスチャ読み取りキュー・バッファは、テクスチャデータブロックに対する要求を読み取り、共用メモリに送る。命令・メモリ制御装置３８０は、共用メモリへのアクセスを判定するが、さらに、メモリからのデータを蓄えるためのバッファをも含む。テクスチャキャッシュ制御装置３９１、テクスチャ読み取りキュー３９３、及び、命令・メモリ制御装置３８０は、図１１のテクスチャブロック・フェッチ・ユニット４２９の特異な実現例である。圧縮されたテクスチャブロックに対しては、圧縮キャッシュ４１６及び復元エンジン４０４が、テクスチャブロック・フェッチ・ユニット４２９の一部となる。テクスチャキャッシュ制御装置３９１は、復元エンジン４０４を介して、圧縮キャッシュ４１６からテクスチャキャッシュ４０２へと向かうテクスチャブロックの流れを管理する。

【０１４１】

走査変換ブロック３９７及びテクスチャフィルタ・エンジン４０３は、図１１のポストラスタライザの特異な実現例である。走査変換ブロック３９７及びテクスチャフィルタ・エンジン４０３は、前述の、図９Ａに示した同類と同じように動作する。

【０１４２】

テクスチャキャッシュ制御装置

テクスチャ・フェッチ動作にとって高いレイテンシを有する環境で、ラスタライズする２つの手法を説明した。次にテクスチャキャッシュ制御装置の特徴を詳細に説明する。

【０１４３】
テクスチャキャッシュ制御装置方式では、ラスタライザはテクスチャ・マップ・フェッチ制御のレイテンシが短いにも関わらずテクスチャ・マッピング中にフル・スピードで機能することができる。タイラーにおいて、このレイテンシは、共用メモリ(例えばＲＡＭＢＵＳ) からの未圧縮データを読み取るのに要する時間と、テクスチャ・マップのブロックを復元するのに要する時間を加算したものである。共用メモリからｇスプライトブロックをフェッチし、可能であればこれを復元し、さらにｇスプライト空間の画素データをビュー空間(具体的には画面座標) に変換するｇスプライト・エンジンに応用される。
【０１４４】
テクスチャキャッシュ制御装置方式は、時間内にオフセットされるテクセル(又はｇスプライト画素) 要求の同一のストリームを２つ作成することを基本的に前提としている。最初の(早いほうの) ストリームはいずれのテクスチャデータも戻されることのないプリフェッチ要求で、また第２の(遅いほう) ストリームはテクセルデータを戻すための実際の要求である。この２つのストリームの時間差を、テクスチャデータの読み込みや復元の際のレイテンシが表れなくなるように用いる。
【０１４５】
上記の時間分離要求を生成させる２つの方法は、(１) ともに単一のプリミティブＦＩＦＯ（図１１及び図９Ｃ）からデータを読み出す一対のラスタライザと、（２）画素ＦＩＦＯ(図１０及び図９Ｂ) がその後に続く単一のラスタライザである。
【０１４６】
（１） の方法では、第１のラスタライザはプリミティブＦＩＦＯの入力側又はその近辺の位置からプリミティブを読み取ってプリミティブのラスタライズを行なうので、テクスチャ要求を作成することができるが、テクセルを受信することや、画素を作成することはない。第２のラスタライザはＦＩＦＯ出力からプリミティブを取り除いて後で同一の要求を作成し、テクスチャキャッシュ制御装置からテクセルを受信し、画素を作成する。各プリミティブ当たりの画素数と組合わされたプリミティブキューの深度が、２つの要求ストリーム間の潜在的な時間差を決める。
【０１４７】
（２） の方法では、単一のラスタライザがプリミティブの処理を行ってテクスチャ要求を作成し、部分的に完全な画素データを画素ＦＩＦＯに出力する。この一部の画素データには、一旦テクスチャ要求が受諾された後で画素の計算を完了させるのに必要なデータ全てが含まれる。画素ＦＩＦＯの出力側には、部分画素が全てあり、これを元に同一のテクスチャ要求のストリームが作成され、テクセルを受信し、完全画素を作成する。画素キューの深度が、２つの要求ストリームの潜在的な時間差を決める。
【０１４８】
テクスチャキャッシュ制御装置
テクスチャキャッシュ制御装置には、仮想キャッシュと物理キャッシュの２つの概念キャッシュがある。仮想キャッシュは最初の(プリフェッチ) 要求ストリームに関連しており、キャッシュエントリに直接伴うデータは有していない(このキャッシュへの要求によりデータが戻ることはない) 。物理キャッシュは第２の(実際の) 要求ストリームに関連しており、各キャッシュエントリに伴う実際のテクスチャデータを有している(従ってデータは要求元に戻る) 。これらのキャッシュのエントリ数は同一である。
【０１４９】
仮想キャッシュは物理キャッシュの後の内容の制御と追跡を行なう。従って、その要求ストリームのいずれの位置でも、物理キャッシュが(後に) その要求ストリームの中の同じ相対位置で有するのと同じ一対のキャッシュキーとエントリを有している。
【０１５０】
要求を受けると(新規の「キー」) 、仮想キャッシュがその現在のキーの組に対して比較を行なう。要求されたキーが仮想キャッシュになければ、キャッシュ置換動作が行われる。仮想キャッシュ置換動作では、１)(ＬＲＵ 又はその他のアルゴリズムによって) 置

換するエントリを選択し、２）そのエントリに対するキーを置換し、３)(メモリと）復元システムを起動してそのキーに関連するデータのフェッチと復元を行なう。図９Ｂ及び図９Ｃに示す特定の方法では、復元サブシステムは、命令・メモリ制御装置３８０、圧縮キャッシュ４１６、復元エンジン４０４から構成される。

【０１５１】

復元サブシステムの出力は、後に物理キャッシュのエントリ(例えばテクスチャキャッシュ４０２）の内容となるテクスチャデータのブロックである。図９Ｂ及び図９Ｃに示すタイラーにおいて、復元サブシステムが行なう処理は順序が保たれているマルチエントリパイプラインで実行される。

ちなみに要求されたキーがすでに仮想キャッシュに存在していた場合、関連データは第２の要求ストリームから要求があった時には物理キャッシュに存在しているので何も実行されない。

【０１５２】

物理キャッシュに要求があると、同様のキーの比較を行って要求されたデータがすでにキャッシュに存在しているか確認する。同一のキーが見つかると、関連データが戻される。同一のキーがない場合は、復元サブシステムが出力する次のデータが確実に所望のデータとなる。なお、この新規データに置換される物理キャッシュは、仮想キャッシュ制御装置で計算されたキャッシュエントリの「目標」インデックスに基づき仮想キャッシュから命令を受け、要求されたデータと共に復元サブシステムを通過する。

【０１５３】

この方式を適切に機能させるには、復元サブシステムと物理キャッシュとの間のインターフェイスに対してフロー制御を行なう必要がある。復元されたデータをすぐに物理キャッシュの中の目的のエントリに上書きすることが可能な場合、そのキャッシュエントリの以前の内容に関連するもの全てが完全にそろっていないことが考えられる。(ちなみに、物理キャッシュ制御装置も復元サブシステムがデータを出力するまで待機しなければならない。)

【０１５４】

前のエントリの内容を上書きする前に新しいエントリが要求されるまで待機することによって、このフロー制御は行われる。テクスチャキャッシュへの新規データ書き込みは、それが必要となる最後の瞬間まで常に延期される。

この置換は必要となるときまで延期されるので、物理キャッシュへのデータ書き込みに要する時間により、レイテンシは第２の要求ストリームを制御する処理に加算される。このレイテンシを低減させる２つの方法は次の通りである。

最初の方法では、物理キャッシュのバッファデータを２倍にする。これにより、復元サブシステムはそのダブルバッファ側に各エントリのデータを即座に書き込み、また物理キャッシュ制御装置はバッファスワップ(通常、高速)を行ってデータをそのキャッシュ側にマップさせることが可能になる。書き込まれるエントリに空きがなく、まだスワップされていない場合は、復元サブシステムは待機するのみである。ちなみに、仮想キャッシュ制御装置が使用するキャッシュ置換アルゴリズムは同じエントリに繰り返し書き込みを行なう傾向があるので、キャッシュエントリへの書き込みを「拡張」する必要がある。

【０１５５】

第２の方法では、複数の「キー化した」エントリの他に１つ又は複数の「追加」エントリを物理キャッシュに設ける。キー化したエントリの数は、存在するキャッシュキーの数と同じであり、仮想キャッシュのエントリ数と一致する。追加エントリの数は、アンマップされた(即ち、現在データの書き込まれていない）エントリの数を表している。このようなエントリの総数は、物理キャッシュのデータエントリの総数となる。

第２の方法では、キャッシュエントリは全て(キーに関連して）何度もアンマップ又はマップされる。アンマップされたエントリの集合は、復元サブシステムがデータの完全ブロックを書き込むエントリのＦＩＦＯを形成している。分離ＦＩＦＯ構造は、このようなアンマップされたエントリに関連する目標インデックスのために維持される。物理キャッ

シュへの要求があったが一致するキーがない場合は、アンマップされたエントリのキューの最初のエントリが目標となるインデックスにマップされてそのキーに対応させられる。置換されたエントリはアンマップされ、アンマップ状態のキューの最後に配置される(空)。

【０１５６】

キャッシュキー生成

この方式は同一の要求ストリームが２つ生成されることを基本的に前提としている。しかしながら、このような要求に対応する特定のキーは同一である必要はない。

第１の(早いほうの) 要求ストリームを形成するキャッシュキーは、テクスチャデータの読み出しとその後の復元を制御するのに使用される。このようなキーは、要求されたデータ(メモリアドレス等) に直接関連していなければならない。

【０１５７】

第２の(遅いほうの) 要求ストリームを形成するキャッシュキーは、最初のストリームの内容と正確に一致する必要はない。両者の間に他と異なる１対１のマッピング関係が成立していることのみ必要となる。これは第２のストリームのキーが既存のキャッシュエントリのマッチングのためだけに使用され、データフェッチ動作には使用されないためである。ここにおいては、新しいデータが物理キャッシュにマップされたときに物理キャッシュのキーとキャッシュエントリとの間の関係が成立し、また対応するエントリのインデックスが仮想キャッシュによって計算されて復元サブシステムを通過するということが重要である。

【０１５８】

このことは、ストリームのキーが他と異なってさえすれば「正確」である必要はないため、第２の要求ストリームキーを生成する処理の制御を簡素化するのに利用できる。

【０１５９】

図１２Ａは画像処理ボード１７４上のｇスプライト・エンジン４３６を示すブロック図である。ｇスプライト・エンジン４３６は収集したｇスプライトから出力されるグラフィックスを生成する。ｇスプライト・エンジン436はタイラー・メモリ・インタフェースユニットとのインターフェイスを確立して、共用メモリの中のｇスプライト・データ構造にアクセスする。ｇスプライトはｇスプライト・エンジンによって変換され(回転、縮小拡大など)、合成バッファに送られて他のｇスプライトから成る画素と合成される。

【０１６０】

インターフェイス制御部４３８は、タイラーを介してｇスプライト・エンジンと共用メモリとの間にインターフェイスを確立するのに使用される。このブロックはＦＩＦＯを有しており、メモリからのアクセスがｇスプライト・エンジンで分散される前にバッファする。

【０１６１】

表示制御処理装置４４０は、ビデオ表示の更新制御に使用される。これは、ビデオ表示のリフレッシュを制御し、またｇスプライトアクセスを制御するのに必要なタイミング信号を生成するビデオタイミングジェネレータを有している。また、このブロックはｇスプライト表示データ構造をトラバースして、所定の３２走査線帯に対して読み出す必要のあるｇスプライトを決定する。

【０１６２】

ｇスプライト・ヘッダ・レジスタ４４２は、画像処理アドレスジェネレータ(ラスタライザ)４５４とｇスプライトフィルタエンジン４５６が使用するＧスプライト・ヘッダを保存して、各ｇスプライトでの変換を決定する。またｇスプライト・ヘッダデコーダ４４４もこのレジスタを使用して、各バンド幅でｇスプライトを与えるに必要なブロック（この場合、８×８圧縮ブロック）を決定する。

【０１６３】

ｇスプライト・ヘッダデコーダ４４４は、各ｇスプライトからのブロックが３２走査線帯で可視状態になるブロックを決定し、ｇスプライト読み出しキュー446に転送されるブ

ロック読み出し要求を生成させる。このブロックも、ｇスプライト・エッジ方程式パラメータを使用してｇスプライトを現在のバンド幅にクリップする。この処理を以下に詳しく説明する。

【０１６４】

ｇスプライト読み出しキュー４４６はｇスプライトブロックの読み出し要求をバッファする。本実施例では、このキューは１６個のブロックの要求を記憶している。

ｇスプライト・データアドレスジェネレータ４４８は要求されたｇスプライトブロックのメモリのアドレスを決定し、ｇスプライト読み出し要求をインターフェイス制御部に送信する。ｇスプライト・データアドレスジェネレータ４４８はメモリ管理ユニットを有している。

【０１６５】

共用メモリ２１６（図４Ａ）から取り出した圧縮データは、圧縮キャッシュ４５８に一時的に保存できる。

復元エンジン４５０には２つのデコンプレッサを備えており、一方は三次元のｇスプライトや画像等の連続したトーンの画像に対してＤＣＴに基づいたアルゴリズムを実行し、他方はデスクトップ画素データに対してロスレス・アルゴリズムを実行する。ＤＣＴに基づいたアルゴリズムは２つの並行した圧縮ブロックによって実現され、双方のブロックとも、各々のクロック周期ごとに８個の画素要素（即ち、２画素）を生成することができる。

【０１６６】

ｇスプライト・キャッシュ４５２は、８×８の１６ブロックに対して復元されたｇスプライト・データ（ＲＧＢ∝）を保存する。データは１６個のｇスプライト画素がクロック周期ごとにアクセスされるよう構成される。

画像処理アドレスジェネレータ(ラスタライザ)４５４は、特定のアフィン変換に基づいて各ｇスプライトを走査し、また各画素のフィルタパラメータを計算するのに使用される。ｇスプライト・キャッシュアドレスはｇスプライト・キャッシュ４５２のｇスプライト・データにアクセスし、それをｇスプライトフィルタエンジン４５６に送るために生成される。画像処理アドレスジェネレータ(ラスタライザ)４５４も合成バッファを制御する。

【０１６７】

ｇスプライトフィルタエンジン４５６は、フィルタパラメータに基づいて画素位置の画素のカラーとアルファを計算する。このデータは合成バッファに転送されて合成される。このブｇスプライトフィルタエンジン４５６は、画素位置でのｇスプライト ｓ－ｔ座標に基づいて４又は１６画素フィルタ・カーネルを計算する。例えば、フィルタはバイ・リニア又はさらに優れた余弦の和の関数でよい。１６画素フィルタ・カーネルは、バイ・リニア補間法よりも細かなフィルタリングが可能なネガティブローブを有することができる。ｇスプライトフィルタエンジン４５６はクロック周期ごとに合成される４つの新しい画素を生成する。これらの画素は２×２パターンで配列される。

【０１６８】

ｇスプライト・エンジン４３６はタイラー２００と合成バッファ２１０とのインターフェイスを確立する。制御信号はビデオタイミングとＤＡＣ２１２へのデータ転送を制御する。

図１２Ｂはｇスプライト・エンジン４３６の他の構成を示すブロック図である。これには、プリラスタライザ４４９と画像処理アドレスジェネレータ(ラスタライザ)４５４の双方が含まれているので、ｇスプライト・エンジンはｇスプライト画素データのブロックの検索や復元でレイテンシを生じさせることなくｇスプライト空間からの画素データを画面空間に変換することができる。この構成で２つのラスタライザを用いる方法は図１１及び９Ｃを参照して上記に説明したとおりである。

【０１６９】

ｇスプライト・エンジン４３６におけるブロックの動作は、２つのラスタライザを使用してテクスチャデータのブロックをフェッチする方法を採用している点以外は、図１２Ａ

で説明したものとほぼ同じである。この構成(図１２Ｂ)では、ｇスプライト・ヘッダデコーダ４４４がｇスプライト・ヘッダ・レジスタ４４２を読み出し、ｇスプライトを現在の表示バンドにクリップし、さらにｇスプライトキュー４４７にｇスプライトを移してラスタライズを行なう。データアドレスジェネレータ又は「プリラスタライザ」４４９はｇスプライト・ヘッダの所定のアフィン変換式に基づいて各ｇスプライトを走査し、ｇスプライト・キャッシュ制御部４５１に対して読み出し要求を発する。テクスチャキャッシュ制御装置に関連して上記に説明した方法を用いれば、画像処理部４５５が、ｇスプライト・データのブロックを必要とするときに、その要求されるｇスプライト・データが、ｇスプライト・エンジン４３６にあり、且つ、特にｇスプライト・キャッシュ４５２にあることを、ｇスプライト・キャッシュ制御部４５１によって確実にする。この制御は、復元エンジン４５０を介しての圧縮キャッシュ４５８からｇスプライト・キャッシュ４５２へのｇスプライト・データのブロックの流れを管理する。読み出しキュー４５３は共用メモリシステムへのｇスプライト・データブロックの要求をバッファする。またインターフェイス制御部４３８は読み出しキュー４５３の要求を読み出し、共用メモリへのアクセスを制御し、ｇスプライト・データブロックを圧縮キャッシュ４５８に入れる。

【０１７０】

ｇスプライト・エンジンの復元サブシステムは圧縮キャッシュ４５８と復元エンジン４５０を有している。ｇスプライト・キャッシュ制御部４５１は、テクスチャキャッシュ制御装置に関連して上記に説明したこの復元サブシステムを介してｇスプライトブロックの流れを制御する。

【０１７１】

画像処理アドレスジェネレータ(ラスタライザ)４５４は、ｇスプライト・ヘッダの所定のアフィン変換式に基づいて各ｇスプライトを走査し、各画素のフィルタパラメータを計算する。また、ｇスプライト・データのｇスプライト・キャッシュアドレスも生成し、これをｇスプライトフィルタエンジン４５６で使用できるようにｇスプライト・キャッシュのキャッシュアドレスマップへ送る。キャッシュ構成の一例では、キャッシュアドレスマップはアクティブとなる１４画素ブロックと復元エンジンから書き込みされる２つのブロックを選択する。

【０１７２】

ｇスプライトフィルタエンジン４５６は、ｇスプライト空間の画素位置にあるカラー及びアルファデータを画面空間にマップする。この方法では、２ｘ２又は４ｘ４のフィルタ・カーネルを用いて、画面空間の画素位置にある画素値(カラー又はカラーとアルファの双方)を計算する。合成バッファ制御部４５７は、画素値、この場合は１クロック周期当たり４画素を合成バッファに送る。合成バッファ制御部４５７は合成バッファからのレディラインを監視し、ｇスプライト・エンジン４３６が合成バッファをオーバーランさせないようにする。画像処理アドレスジェネレータ(ラスタライザ)４５４は合成バッファ制御部４５７を制御する。

【０１７３】

図１３は、画像処理ボード１７４の合成バッファ４８０を示すブロック図である。合成バッファ４８０は、ｇスプライト・エンジンからのｇスプライト・データを合成してＤＡＣ２１２へ転送するディジタルビデオデータを生成するのに使用する専用メモリ機器である。合成バッファは一度に32本の走査線で動作する。１つの３２走査線帯でｇスプライトを合成する間に、前の３２本の走査線が表示される。

【０１７４】

合成ロジック４８２は、走査線バッファに書き込む画素値を計算する。これは、現在走査線バッファに保存されている画素値と合成バッファに書き込まれている画素値とを混合させることで実行される。この操作は以下に詳しく説明する。一構成例では、合成ロジックが各クロック周期で画素についての４つの並行動作を実行する。

【０１７５】

メモリ制御部４８４はアドレスの制御とメモリバンクのサイクリングに使用される。ア

ドレス情報は、通常のＤＲＡＭを使用するときのように行と列のフォーマットで送られる。

アルファ・バッファ４８６は、１３４４×３２画素のそれぞれに対して８ビットの数値を有している。メモリは、連続する４つの画素が各クロック周期で読み出し、又は書き込みされるように構成されている。アルファ・バッファは、３２走査線帯の切り替えの間にバッファを迅速にクリアする高速クリア・メカニズムも有している。

【０１７６】

２つの独立した走査線バッファ４８８が使用される。走査線バッファは１３４４×３２画素のそれぞれに対して３種類の８ビットカラー値を有している。メモリは、連続する４つの画素が各クロック周期で読み出し、又は書き込みされるように構成されている。一方のバッファは、あるバンドの画素データをＤＡＣに転送するのに使用し、他方のバッファは、次のバンドの画素を合成するのに使用する。そのバンド処理が一旦終了すると、その機能はスワップされる。

マルチプレクサは２つの走査線バッファ４８８のうちいずれかからのデータを選択し、画素表示データをＤＡＣに送信する。マルチプレクサは３２本の走査線ごとにバッファの切り替えを行なう。

【０１７７】

合成バッファ４８０は、ｇスプライト・エンジン２０４とのインターフェイスを確立し、画像データをＤＡＣ２１２へ転送する。

図１４は画像処理ボード１７４上のＤＡＣ ５１４を示すブロック図である。ＤＡＣ ５１４は、現在市場に一般的に出回っているＲＡＭＤＡＣに共通する基本機能を実行する。ＤＡＣ は内部制御レジスタを読み書きし、ビデオ制御信号をパイプライン処理するロジックを有している。その他の機能ブロックについては、以下に説明する。

【０１７８】

画素データルーティングブロック５１６は合成バッファからの画素データのルーティングを制御する。通常の動作モードでは、このデータは画素速度で３つのチャネルのそれぞれのカラー ＬＵＴ ５１８に送られる。このブロックもＤＳＰからデータを再び読み出して診断することが可能である。

ステレオ画像スプリッタ５２０は、ヘッドマウント表示システムを用いて立体表示される２つのビデオ信号を支援する。このモードでは、２つのビデオチャネル(５２２，５２４）が合成バッファから交互配置されており、ＤＡＣ ５１４によってスプリットされなければならない。ステレオ画像スプリッタ５２０はＤＡＣ ５１４でこの機能を実行する。通常の単一チャネルモードでは、ＬＵＴデータが第１ＤＡＣ に直接送られる。

【０１７９】

その他、単一のビデオ出力を生成するようＤＡＣ ５１４を構成することも可能である。ＤＡＣ は、単一のビデオ出力と共に、走査線インターリーブフォーマットを使用して立体表示を作成することが可能である。この場合、一方の目に対する走査線に、他方の目に対する走査線が続く。結果として生成されたビデオストリームは６４０ｘ９６０等のフォーマットを有しており、例えばこれは６４０ｘ４８０の画像を表している。

【０１８０】

クロックジェネレータ５２６はビデオクロックとオーディオクロックを生成する。これらのクロックは２つの位相固定のクロックジェネレータによって生成され、同期ずれを除去する。クロックジェネレータもメディアチャンネルからの制御信号に従い、画像処理ボードを外部同期ソースに同期させることが可能である。

【０１８１】

本システムの構造と動作を上記に説明したが、次にシステムの構成部分と特徴について詳細に説明する。まず、上記に説明した概念を実行するためにシステムで使用されるデータ構造を説明する。

【０１８２】

チャンキング

ＲＡＭ に大容量のフレームバッファとZバッファを使用してカラー値、深度値、各画素のその他の情報を記憶する従来のグラフィックシステムとは異なり、このシステムはある１画面の中のオブジェクトを「チャンク」と呼ばれる画像領域に分割し、オブジェクト幾何形状をこのようなチャンクに個々にレンダリングする。実施例において、オブジェクトはｇスプライトで表示される。ｇスプライトはチャンクにサブ分割され、チャンクは別個に表示される。この説明は幾つかの特定の実施例について言及しているが、チャンキングは本発明の範囲から逸脱せずに様々に応用することができる。

【０１８３】

チャンキングの概念を、図例を挙げて説明する。図１５Ａに示すように、境界ボックス５４８と呼ばれるボックスで囲まれているグラフィック画面のオブジェクト５４６は、ｇスプライト５５２という画像領域に表示される。境界ボックスを回転、拡大・縮小、拡張、その他の変換(例えばアフィン変換) を行い、画面領域にｇスプライトを作成してもよい。境界ボックスが一度作成され、境界ボックスが３２画素境界(即ち、チャンク境界) ５５４に収まらないのであれば、境界ボックスをオブジェクトの周囲でＸ及びＹ方向に拡張して３２画素チャンク・サイズの整数倍になるようにする。図１５Ｂのオブジェクト５５０から分かるように、図１５Ａに示すオブジェクト５４６を囲む境界ボックスは図１５Ｂの３２×３２画素境界に拡張される。ｇスプライトはレンダリングされる前に３２×３２画素「チャンク」５５６にサブ分割される。しかし、チャンク・サイズはこれよりも小さい又は大きいものを用いることもできるし、また他の形状のチャンクも使用できる。しかし、矩形、より好ましくは四角形のチャンクを図示する。

【０１８４】

図１５Ｃに示すように、グラフィック画面５８１は重なり合ったオブジェクト(５６０，５６２）を複数有している。これらのオブジェクトは境界ボックスで囲まれており、ｇスプライト(５６４，５６６）に割り当てられる。図１５Ｃの境界ボックスは３２画素倍にすでに拡張（及び回転、縮小拡大、その他の変換）されており、３２×３２チャンク５６８が生成される。しかし、図１５Ｃから分かるように、ｇスプライトとそれに対応する３２×３２画素チャンク境界５７０は３２画素画面境界に正確に配列されないので、チャンクを行っている間にさらにｇスプライトを操作してｇスプライトが画面空間に収まるようにしなければならない。

【０１８５】

チャンキングを利用してレンダリングされるｇスプライトを作成する１つの方法は、個々のオブジェクト幾何形状を含む小さな個々のｇスプライトを複数作成及びレンダリングする代わりに、複数のオブジェクトを組み合わせてより大きな合成ｇスプライトを作成することである。ｇスプライトを組み合わせるとレンダリングの際の処理時間が短くなり、また組み合わせたオブジェクトがグラフィック画面の中で頻繁に変わらないのであればこの方法は極めて好ましい。ｇスプライトを作成する別の方法は、複雑な幾何形状を有するオブジェクトの成分をターゲットとし、これらの複雑な幾何形状成分を複数のｇスプライトにサブ分割することである。このようにサブ分割を行なうと処理時間が長くなってしまうおそれがあるが、頻繁に変化する特定の複雑なオブジェクトの出力解像度を向上させることができる。オブジェクトによっては、これら技術を組み合わせて使用してもよい。

【０１８６】

例として、腕を様々なサイズの複数のスパイクでカバーされているビデオゲームのキャラクタを挙げる。この場合、腕は頻繁に動くものとする。キャラクタの体、頭、その他の部分は頻繁には動かないので、より大きな合成ｇスプライトに組み合わせることができる。しかし、スパイクでカバーされており、複雑な幾何形状となり、また頻繁に動くキャラクタの腕は、出力解像度を向上させるために複数のｇスプライトに分割される。この場合、そのような組み合わせとサブ分割の双方を用いる。このようなキャラクタを図示するのは容易ではなく、また実用的ではないので、より単純なオブジェクトである「コーヒーカップ」を用いてそのような組み合わせとサブ分割を図示する。

【０１８７】

図１６Ａは「コーヒーカップ」を表している。この「コーヒーカップ」は実質的に複数の別個のオブジェクトから構成されている。例えば、「コーヒーカップ」は実際にカップ容器と、カップ取っ手と、ソーサーと、カップから立ち上る湯気とで構成されているように見える。図１６Ａに示すように、１つの方法は個々のオブジェクトを組み合わせてより大きなｇスプライト(即ち、「コーヒーカップ」) にするものである。もう１つの方法は、図１６Ｂに示すように、「コーヒーカップ」を複数の小さなオブジェクト(例えばカップ容器、カップ取っ手、ソーサー、湯気) にサブ分割してそれぞれ小さなｇスプライトを作るものである。また図１６Ｂは複雑な配置のオブジェクトをサブ分割する方法も示している。

【０１８８】

図１６Ａに示すように「コーヒーカップ」５７４を１つの単純なオブジェクトとすると、オブジェクトの各構成部分(例えばカップ容器、カップ取っ手、ソーサー、湯気) を組み合わせて１つの大きなｇスプライトを作成することができる。この場合、境界ボックス５７６でオブジェクトの周辺を囲み、オブジェクトを画面空間に変換して１つの大きなｇスプライトを作成する。画面空間の３２×３２画素の境界内に収まるｇスプライトを作成するために、境界ボックスを回転、拡大・縮小、拡張、その他の変換をさせてもよい。その後、ｇスプライトは複数の３２×３２画素チャンク５７８に分割される。

【０１８９】

ｇスプライトをチャンクに分割する１つの方法は、オブジェクトに含まれる幾何形状を全てつなぎ合わせてチャンクにその幾何形状を置くというものである。別の方法では、問題のチャンクを使用する幾何形状の全てを記録しているチャンクをつなぎ合わせる。図に示す実施例では後者の方法を使用しているが、前者やその他の方法も使用できる。図１６Ａから分かるように、複数のチャンクが空の状態になる(即ち、いずれのオブジェクト幾何形状もチャンクを使用していない) 。後で説明するが、これらのチャンクはレンダリングが行われている間は無視される。

【０１９０】

次に、「コーヒーカップ」を１つの複雑なオブジェクトとした場合、図１６Ｂに示すようにオブジェクトは処理されて複数の小さなｇスプライトを作成するより小さなオブジェクト構成部分にサブ分割される。例えば、オブジェクトの「コーヒーカップ」は、取っ手のないカップ容器５７９と、カップ取っ手５８０と、ソーサー５８１と、湯気５８２というサブオブジェクトから構成される。各下位オブジェクトはそれぞれ５８３－５８６に示すように境界ボックスで囲まれて、４つのｇスプライトを形成する。４つのｇスプライトを含む「コーヒーカップ」も、５８７で示すように境界ボックスで囲まれる。画面空間の３２×３２画素境界に収まるｇスプライトを作成するために、各境界ボックスを回転、縮小拡大、拡張、その他の変換(例えば、アフィン変換) させてもよい。次に各ｇスプライトは３２×３２画素チャンクに分割する。境界ボックス５８７もチャンクに分割させて、レンダリングの際に無視される空のチャンク５８８を有するようにさせる。しかし、この境界ボックスのチャンクは図１６Ｂに図示されていない。

【０１９１】

チャンキングにより、グラフィック画像を単一のフレームとしてレンダリングするわけではなく、後にフレーム又はビュー空間に統合される一連のチャンクとしてレンダリングする。現在描かれている画像の３２×３２画素チャンクを交差する単一のｇスプライト内のオブジェクトのみをレンダリングする。チャンキングにより、フレーム及びＺバッファはメモリの小さな物理サイズとでき(即ち、上記に説明した従来のグラフィックシステムよりもはるかに小さなメモリを使用できる) 、使用されるメモリの使用頻度が上がり、メモリのバンド幅が増大する。また、チャンク・サイズが小さいことは、フレーム及びＺバッファが大きいと効率的に応用することができないような、さらに高度なレンダリング技術を使用することができるようになる。

【０１９２】

チャンクのレンダリングはタイラーで実行される。しかし、レンダリングは他のハード

ウェア構成要素上、又はソフトウェアを使用しても実行できる。タイラーチップのＶＬＳＩメモリは、現在レンダリングされているフレームの小さなチャンク(３２×３２画素)を保存するのに使用される。オンチップＶＬＳＩメモリは、外部ＲＡＭよりも極めて高速でメモリバンド幅も広い。しかし、チャンキング処理が行われるために、フレームバッファ及びＺバッファ全体を保存する大量のメモリは不要になる。タイラー内の内部メモリは現在のチャンクの処理のみを行い、後続するチャンクの各々を処理するために繰り返し使用される。そのため、使用可能な内部メモリはグラフィックレンダリング中に頻繁に使用される。

【０１９３】

また、内部ＶＳＬＩメモリを使用すると、オフチップ通信や、従来のフレーム及びＺバッファに必要な大容量外部メモリに対する読み出しや書き込み動作によるオーバーヘッドが原因で、通常生じるピンドライバの遅延を除去することができる。また、図に示す本実施例では後続するチャンクが計算される前に３２×３２画素チャンク全体が完全にレンダリングされるので、チャンク・サイズが小さいと、外部メモリの大部分に保存されるフレーム及びＺバッファ全体よりもチャンクのほうがより高度なアンチエイリアシング(例えばフラグメントバッファ) 及びテクスチャリングを実行することができる。さらにチャンク・サイズが小さい場合、後で詳細に説明する画像圧縮技術を使用する際に便利である。

【０１９４】

チャンク及び解析されたフラグメントに、交差するポリゴンを全て描出した後、カラーや不透明度を含む画素データが、タイラーチップにおいて圧縮されて、外部メモリに転送される。

【０１９５】

図１７Ａ及び１７Ｂはグラフィック画面のチャンクへの分割方法の概略を詳細に示している。まず、１つ以上の境界ボックスが各オブジェクトに対して作成される（５９２)(図１７Ａ)。オブジェクトが複雑な幾何形状(例えば細かい碁盤目状など)を有する場合（５９４）、複数の境界ボックスが作成されて(複数のｇスプライトを作成するために)、オブジェクトの複雑な各構成部分を囲む(５９６)。オブジェクトが複雑な幾何形状を有しない場合は、単一の境界ボックスを使用してオブジェクトを囲み、ｇスプライトを作成することができる(５９８)。しかし、オブジェクト幾何形状が複雑であれば、単一の境界ボックスが、オブジェクトの複雑な成分を囲むために生成された複数の境界ボックスをも囲むことになる。単一又は複数の境界ボックスが、32画素の整数倍でないならば(６００）、このような境界ボックスを３２画素の整数倍になるようＸ又はＹ方向(又は双方向) に対称に拡張する。次に、オブジェクト(幾何形状が複雑なときはオブジェクト成分)を境界ボックスでセンタリングする(６０２）。これを図１５Ｂ及び１５Ｃのｇスプライトで示す。拡張は対称に行なうのが好ましいが、必ずしもそうする必要はない。拡張が対称であると、単一のｇスプライトにおけるチャンク間の処理のバランスが良くなる。

【０１９６】

再び図１７において、ｇスプライトは３２×３２画素チャンクに分割される(６０４)(図１７Ｂ)。明らかなように、チャンクはビュー空間の固定位置にあるのではなく、チャンクされたオブジェクトの位置によってはアドレス呼び出しが可能な可変位置に置かれる。ｇスプライトをチャンクに分割した後、チャンクの処理を行なう。チャンクのレンダリングが完了すると(６０６)、処理は終了する。チャンクのレンダリングが完了しない場合、まず後続するチャンクが空なのか確認した後このチャンクの処理を始める(６０８)。チャンクが空の場合、処理は行われず、後続するチャンクの確認が行われる。チャンクが空でない場合は、チャンクを使用しているオブジェクト全てが処理されるまでチャンクのレンダリング(６１０) をタイラーで続ける。この処理は、各ｇスプライトの全チャンクと全ｇスプライトの処理が完了するまで続けられる。

【０１９７】

ｇスプライトのサイズは画面の総領域のパーセンテージで表される。背景のｇスプライトは極めて大きいが、通常、画面の他の構成部分は画面の総領域よりも小さい。チャンキ

ングの性能はｇスプライトのプリミティブの画面空間のサイズによって変わる。そのため、ｇスプライトの作成に使用するオブジェクトのデータ入力ストリームを適切に調整(例えばキュー) する必要がある。オブジェクトデータ入力ストリームを適切に調整すると、オブジェクトの処理をより高いバンド幅で実行することができ、またシステムのスループットも上昇する。

【０１９８】

このシステムはコマンドストリームキャッシュを使ってオブジェクトのデータ入力ストリームをキャッシュしている。コマンドストリームキャッシュを使用してｇスプライトの全内容をキャッシュし、また全てのチャンクとそれに対応するキャッシュに保存されているｇスプライトの関連する幾何形状を繰り返し使用することができる。

またキャッシュは選択キャッシングにも使用できる。例えば、幾何形状プリミティブが所定数のチャンクを使用するときに、これらのプリミティブが自動的にキャッシュされるよう閾値を決める場合である。キャッシュの使用が可能であれば、仮想チャンキングを行なうことができる。仮想チャンキングでは、それぞれが仮想チャンクであるＮ×Ｍ チャンクの領域に相当するチャンクバケットを作成する。仮想チャンキングにより、ｇスプライトの内容と幾何形状のサイズに合った仮想チャンクの適応サイズ設定を行なうことができる。

【０１９９】

キャッシュは修正画面・グラフキャッシングにも使用できる。画面の静的部分をキャッシングして参照するのではなく、キャッシングを行って間接的に画面の動的部分を参照する。例えば、やや複雑な幾何形状のカッコウ時計を含むｇスプライトについて考えてみる。時計そのものは極めて複雑であるが、唯一動く部分は鳥、２つのドア、２本の針である。さらに、各幾何形状は、固定されており変化することはない。従って、時計をレンダリングすると、６本の木は静止状態で、また６つの部分が変換される(即ち、時計、鳥、２つのドア、２本の針)。キャッシュの容量が十分大きければ、画面グラフ全体をコマンドストリームに変換する。レンダリングの際には、現在の変換部分はキャッシュされたコマンドストリーム上にパッチされ、その後コマンドストリームはｇスプライトの全チャンクに対して発せられる。コマンドストリームのパッチ部分のサイズはレンダリングしても変わらない。さらに順応性ある方法としては、キャッシュした静止画面グラフに呼び出しコマンドを挿入するというものがある。レンダリングの際は、動的部分が可変サイズのメモリに書き込まれ、キャッシュされる。このような動的部分のアドレスは、静止コマンドストリームの対応する呼び出しコマンドにパッチされる。この方法は、動的コマンドのサイズをレンダリングごとに変えることができるため、順応性に優れている。従って、この場合はメモリにキャッシュされたコールバック方法が効果的である。カッコウ時計の例では、６つの変換部分を書き込んで場合によっては鳥の幾何形状を呼び戻すので、ドアが閉じられる場合に鳥部分は空になる。この方法は、バスのバンド幅に対しては極めてコンパクトで、画面グラフを迅速に指定方向に移動することが可能になる。

【０２００】

キャッシュメモリに制限があっても、幾何形状や属性によっては何度レンダリングしてもキャッシュされた状態を保つことが可能なものもある。例えば、カーレーシングゲームの場合、車体の幾何形状をキャッシュすると、全体的に時間を大幅に節約することができる。同様に、共通の属性状態(又はサブ属性の状態) は複数のｇスプライト又は単一のｇスプライトをレンダリングする際に何度も利用される。上記に述べたように、チャンキングでキャッシュを使用すると、時間が大幅に節約される。しかし、ｇスプライトの各使用チャンクに対してコマンドストリームを迅速に作成することにより、コマンドストリームをキャッシュすることなしで適切なチャンキング性能を実現することができる。

【０２０１】

図９Ａ～９Ｃに示すタイラーの実現例では、複数のチャンクを並行処理装置で同時に使用して計算負荷を共有するのではなく、チャンクを順次使用して１つの処理装置上でフレーム全体をレンダリングする。あまり好ましくはないが、チャンクの逐次又は並行処理を

組み合わせて行なうことも可能である。チャンクを完全並行処理する場合、画面上を移動するオブジェクトに対してはそれが移動したときに固定のチャンキング処理を行なうことが必要になる。しかし図に示す本発明の実施例においては、チャンクを逐次処理しているので、オブジェクトはｇスプライトのチャンク境界に固定される。従ってオブジェクトが画面上を移動してもチャンキングを行なう必要はない。またチャンクを並行処理でレンダリングしても、チャンクを逐次的にレンダリングする場合のような、高度なアンチエイリアシング及びテクスチャリング方法を各チャンクに応用することはできない。３２×３２画素チャンク全体は後続するチャンクが計算される前にレンダリングされて即座に圧縮されるので、チャンク・サイズと順次レンダリングは画像圧縮技術にとって極めて重要である。

【０２０２】

画像の圧縮は、少ないデータで画像を表示し、保存コスト及び／又は送信時間及び送信のコストを減らすために行なう。適切な方法で画像を再構成できる場合、画像を表示するのに必要なデータが少ないほどコストをより下げることができる。また、元の画像を正確に再生するのではなく、元の画像に近づけることでさらに効果的な圧縮を行なうことができる。圧縮量が多くなると、最終画像もより元の画像に近づく(「損失の多い圧縮」)。

【０２０３】

チャンキングの処理そのものは圧縮技術である。オブジェクトは複数の３２×３２画素チャンクから作成される１つ以上のｇスプライトを使って元の画像に接近させられる。実際のオブジェクトはｇスプライトを使って元の画像に近づけ、レンダリングされたｇスプライトから再構成される。元のオブジェクトの再構成は、オブジェクトをｇスプライトに分割してチャンキングすることでどれほど効果的に元の画像に近づけたかによって変わる(例えば、上記の複雑なオブジェクト幾何形状の分割技術)。

【０２０４】

各３２×３２チャンクも画像圧縮技術を使って圧縮される。圧縮された３２×３２画素チャンクは、使用可能な内部メモリの小さな領域の中であまり多くの空間を占領することはない。３２×３２画素チャンクは、離散コサイン変換(ＤＣＴ)を使用する画像圧縮技術で共通して使用されるサイズである１６個の８×８画素チャンクに分割される。

【０２０５】

ある方法では、タイラーの圧縮及び復元エンジン、及びｇスプライト・エンジンの復元エンジンは損失の多い圧縮／復元形式とロスレスの圧縮／復元形式の双方をサポートしている。損失の多い形式には、ＲＧＢからＹＵＶへのロスレスのカラー変換、ＤＣＴ、均一又は知覚量子化、エントロピーコーディング(ランレングス及びハフマンコーディング)がある。ロスレス形式には、ＲＧＢからＹＵＶへのカラー変換、予測ステージ、損失の多い形式でも実行されるエントロピーコーディングがある。

【０２０６】

メモリの必要性を大きく減らしてチャンキングによってグラフィック画像を処理するために、図示の実施例では小さなＺバッファを使用している(例えば４キロバイト(ｋｂ))。特に、この方法でのＺバッファは４ｋｂよりもわずかに小さいが、精度のビット数を変化させることが可能である。しかし、他の大きなサイズ又は小さなサイズのＺバッファも使用できる。小さな４ｋｂのＺバッファを使用すると、１０２４画素のみがＺバッファで一度にレンダリングされる。４ｋｂのＺバッファを使用して任意サイズの画面(例えばｇスプライトで構成される画面)をレンダリングするためには、所定サイズの３２×３２画素のチャンクに分割される(通常、１画面にはｇスプライトが複数あるが、各ｇスプライトともチャンクに分割される。) この方法では、画像前処理装置１０４が適切な幾何形状をｇスプライトの各チャンクに送ってＺバッファでレンダリングを行なう。

【０２０７】

チャンクの機能の例として、図１８Ａに示す８個のオブジェクトとそれに対応する幾何形状を挙げる。簡単に説明するために、８個のオブジェクト６１２～６１９をＡ～Ｄの４つの値のいずれかを有する単一の属性６２０(例えばカラー)で規定する。図１８Ｂに示

すように、８個のオブジェクトはグラフィック画面で重なり合う。各ｇスプライトとその生成については無視し、図示の関係上、分離した４つのチャンクのみについて説明する。この分離した４つのチャンク６２１～６２４を図１８Ｂに示す。分離した４つのチャンクは、図１９Ａに示すように幾何形状１～８と属性Ａ～Ｄに接している。チャンク１６３０(図１９Ａ) は幾何形状１，２，５と属性Ｂに接しており、チャンク２６３９は属性ＡからＤに接しているがいずれの幾何形状にも接していない。またチャンク３６３２は幾何形状２，４，７，８と属性Ａ，Ｂ，Ｄに、チャンク４６３４は幾何形状４，６と属性Ａ，Ｃに接している。(図１８Ｂ及び１９Ａに示すチャンクを使って) 画像の前処理によって作成した部分画面グラフの例を図１９Ｂに示す。各チャンクの属性(例えばＡ～Ｄ，Ｘで示すカラーなど) は円６３８で表し、また属性(例えば１～８で示す様々な形状など) は四角形６４０で表している。文字Ｘは属性のデフォルト値を示している。中間ノードは、幾何形状プリミティブに対して実行する属性演算を有している。画面グラフのリーフノードはチャンクに使用される幾何形状プリミティブを有しており、またチャンクが描く幾何形状を取り囲む境界ボリュームも有することがある(境界ボリュームを有するリーフノードについては以下に説明する) 。

【０２０８】

チャンキングを行なう１つの方法では、各チャンクを繰り返し使用して完全な幾何形状を毎回送信する。さらに最適な別な方法では、現在のチャンクで可視状態の幾何形状のみを送信する(ちなみに、この最適な方法では不明瞭又は目で確認できない幾何形状は省いている)。該システムで使用されている３２×３２画素ブロックのｇスプライトをチャンクする実際の方法は、上記２つの極端な方法の中間に位置するものでバケットチャンキングと呼ばれている。しかし、これら２つの方法に含まれるあるいは中間に位置するような他の方法を用いて、チャンクを作成し、チャンキングを行なうことも可能である。

【０２０９】

バケットチャンキングは２つのパスで構成される。まず第１のパスは画面グラフを移動させながら、現在の変換部分をビュー空間に維持させ、最終的には各チャンクの記述コマンドストリームをビュー空間に構築する。ビュー空間はＮ×Ｍのチャンクバケットに分割され、このチャンクバケットは最終的にはそれぞれのチャンクに対応する幾何形状のリストをそれぞれ有するようになる。幾何形状プリミティブノードがあると、現在の変換部分が境界ボリュームに適用されて２Ｄの「フットプリント」がビュー空間に生成される。フットプリントに接する各チャンクに、幾何形状(及び累積属性状態) が該当するバケットに付加される。この最初のパスの終了時には、各バケットとも該当するチャンクをレンダリングするのに必要なデータを有しているはずである。ちなみに、このチャンキング方法は算出されたフットプリントの品質に左右され、オブジェクトの領域があまり正確でなければ生成されるフットプリントはより大きくなる。従って、周囲を囲まれた幾何形状に接していないチャンクとなる。オブジェクトの領域が正確であると生成されるフットプリントは小さくなるので、ほとんどのチャンクが周囲を囲まれた幾何形状に接することになる。

【０２１０】

第１のパスの例として、図１９Ａに示す幾何形状１～８、属性Ａ～Ｄ，Ｘによって描かれる互いに重なり合うオブジェクトを含む４つのチャンクの部分集合について考察する。パス１の中で画面グラフを移動させる１つの方法として、各チャンクの現在の状態を維持し、所定のチャンクに収まらない幾何形状を省くというものがある。これにより、各チャンクのどの幾何形状の属性内容も最新のものに更新される。図１９Ｂの画面グラフでこの方法を用いると、パス１の後でチャンクバケットに以下のようなコマンドストリームができる。

チャンク１バケット：Ｘ，Ａ，Ｂ，１，２，５，Ａ，Ｘ，Ｃ，Ｄ，Ｃ，Ｘ
チャンク２バケット：Ｘ，Ａ，Ｂ，Ａ，Ｘ，Ｃ，Ｄ，Ｃ，Ｘ
チャンク３バケット：Ｘ，Ａ，Ｂ，２，７，８，Ａ，４，Ｘ，Ｄ，３，Ｃ，Ｘ
チャンク４バケット：Ｘ，Ａ，Ｂ，Ａ，４，Ｘ，Ｃ，６，Ｄ，Ｃ，Ｘ

また、現在の属性状態を維持し、容認された各幾何形状を送信する前にこの状態を送る方法もある。これにより、チャンクバケットに以下のコマンドストリームができる。
チャンク１バケット：Ｂ，１，Ｂ，２，Ｂ，５
チャンク２バケット：＜空き＞
チャンク３バケット：Ｂ，２，Ｂ，７，Ｂ，８，Ａ，４，Ｄ，３
チャンク４バケット：Ａ，４，Ｃ，６
第２の方法は第１の方法を改良したものである。ちなみに属性Ｂは幾何形状２及び５の前に指定された第２及び第３の不要な時間である。この動作は、幾何形状７及び８のＢに対するチャンク３でも行われる。実際には、現在の属性状態がどの属性も各幾何形状に対して再指定されることを表しているので、状況はここに説明する以上に悪くなってしまう。つまり、テクスチャ変換行列が全画面グラフに対して変化しなくても、全チャンクの各幾何形状を送る前に行列が送信されてしまう。

【０２１１】

従って、この特有の方法は、優先属性(overriding attribute)、又はその代わりの構成属性(composing attribute)として、属性を維持する。拡散カラーは優先属性である。画像グラフを作成する画像前処理装置(例えば、画像前処理装置等で実行される画像前処理ソフトウェア) で規定されるように、赤(青(立方体))に使用される属性から赤の立方体ができる。これは、最も近い属性をオブジェクトに結び付ける画像前処理装置のグラフィックインターフェイスとは対称的である。最も近い属性を赤(青(立体))のオブジェクトに結び付けると、青の立方体ができる。
最も遠い属性を優先属性として使用すると、属性維持が極めて簡素化する。画面グラフの移動中に、属性ノードに遭遇すると、画面グラフの中のそのノードよりも下にあるその属性タイプのノード全てを無視することができる。これは最上位の属性が他よりも優先されるためである。
局部変換は構成属性である。従って、現在の値は前の値と新しい値によって規定される。構成属性には、画面グラフが移動して前の数値を保存する際にある種のスタックが必要になる。

【０２１２】

バケットチャンキング方法では以下の構造を用いる：
・　現在の値を有する属性ノード。
・　移動に関するコンテクスト。これは、各優先属性に対して現在の属性値を示すポインタを有する構造である。
・　それぞれコマンドストリームバッファと広域移動に関するコンテクストとして同種のバケットコンテクスト構造を有するバケットのグリッド。
・　それぞれ移動に関するコンテクストによって参照可能なデフォルト属性値のリスト。
初期化するために、コンテクストはデフォルト状態になるので、属性は全てデフォルトコンテクストを表す。デフォルト値は、各チャンクにレンダリング命令を送る前にひとまとめにダンプされるのではなく、ゆっくりとロードされる。

```
Initialize Attribute Maintenance:
for each attribute: attr
        for each bucket: bucket
          bucket.context(attr) ←nil //各バケットのコンテクストをクリア。
        end
        context[attr]←default[attr] // デフォルト値に初期化。
end
```

次に所定の属性ノードの処理方法を説明する。

```
Process Attribute:
if context[attr] ≠default[attr]
        ProcessGem() // 属性が既にセットされ、続く値を無視する。
else
```

```
        Context[attr]←SetAttr(attr,value) // 新たな値にセット。
        ProcessGeom()
        Context[attr]←SetAttr(attr,default[attr])
endif
```

幾何形状ノードを扱う処理により、現在の移動状態が各バケットの属性状態と同期する。

```
Process Geometry:
geomCommand ←ConvertGeometry(geom) // コマンドストリームに変換。
for each touched bucket: bucket
        for each attribute: attr
                if (bucket.context(attr)≠context(attr)
                        bucket.context(attr)←context(attr)
                        append(bucket,context(attr))
                endif
        end
        append(bucket,geomCommand)
        end
```

移動中にスタックを維持するということを除けば、構成属性は優先属性と同様の働きをする。これは、スタック値を保存するためのノードを使用することで可能となる。この方法には以下の構造が必要となる。

・以前の値と新しい値の組み合わせを有する現在の属性ノード。

・移動に関するコンテクスト。これは、全ての構成属性のために、現在の属性ノードを示すポインタを有する構造である。

・それぞれ移動に関するコンテクストによって参照されるデフォルト属性値のリスト。

・それぞれコマンドストリームバッファと広域移動に関するコンテクストと同じタイプのバケットコンテクスト構造を有するバケットのグリッド。

構成属性の初期化は優先属性の初期化と同様と思われる。

```
Initialize Attribute Maintenance:
for each attribute: attr
        for each bucket: bucket
           bucket.context(attr)←nil // 各バケットのコンテクストをクリア。
        end
        context[attr]←default[attr] // デフォルト値に初期化。
end
```

【０２１３】

構成属性の処理を行なうと、移動範囲の中にある現在のノードよりも前に新しい値と全ての数値の組み合わせが作成される。ちなみに値のスタックを作成する場合は、以前の値を保存し元に戻せるようにする必要がある。

```
Process Attribute:
node.ComposedValue ←Compose(context[attr],node.Value)
SavePtr←context[attr] // 前に構成した値を保存。
context[attr]←node
ProcessGeom()
context[attr]←SavePtr //前に構成した値を回復。
```

幾何学処理部は優先属性の場合と同じである。

```
Process Geometry
geomCommand←ConvertGeometry(geom) // コマンドストリームに変換。
for each touched bucket: bucket
        for each attribute: attr
                if (bucket.context(attr) ≠context(attr)
                        bucket.context(attr) ←context(attr)
```

```
                append(bucket,context(attr))
            endif
        end
        append(bucket,geomCommand)
        end
```

バケットチャンキングの第２のパスは、バケットのグリッドを反復させて対応するコマンドストリームを発する。空でない全てのバケットについては、そのバケットに保存されている情報から該当するチャンクがレンダリングされる。ちなみに、画面の中に空のバケットがある場合もあり、これはｇスプライトの全てのチャンクがレンダリングされるわけではないことを意味している。活発に動くｇスプライトのほとんどは、透明な背景上の不透明なオブジェクトから構成され、多数のチャンクが空である必要がある。

【０２１４】

上記の属性状態を維持する方法は、チャンクした状態で幾何形状をレンダリングするのに特に適している。チャンクすることにより幾何形状の集合が、最初に指定されたものとは異なる順序でレンダリングされる。例えば、チャンクのレンダリングにおいて、レンダリングシステムはチャンクと交差しない幾何形状の集合を省く。従って、チャンクした幾何形状のレンダリングのレベルが低い状態では、多くても次の２つの状態レベルを維持するのがよい。１）タイラー又は他のレンダリングハードウェアとの互換性があり幾何形状のレンダリングが可能なフォーマットでの広域状態。２)幾何形状の集合にのみ適用されるその集合内の小域状態でのオーバーレイ。この方法では、幾何形状の各集合は他の集合とは別にレンダリング可能であり、また幾何形状の集合のレンダリングに副次的作用は生じない。

画像圧縮

上記に述べたように、チャンク・サイズと順次レンダリングは画像圧縮技術にとって重要な事である。それは、３２×３２画素チャンク全体が、後続するチャンクの計算が行われる前に完全にレンダリングされ、即座に圧縮することが可能なためである。タイラーは損失の多い圧縮形式とロスレスの圧縮形式の双方をサポートして、チャンクの圧縮を行なう。損失の多い圧縮形式及びロスレスの圧縮形式は共に８×８画素から成る独立ブロックのチャンクを圧縮するので、圧縮された各３２×３２画素チャンクは１６個の圧縮ブロックから構成されることになる。

【０２１５】

画像の圧縮を行なうと、必要なメモリ・サイズが小さくなり、また必要なメモリバンド幅も大幅に減少する。この設計ではキャッシングと、プリフェッチ・ストラレジと、チャンクとを組み合わせて利用することにより、圧縮とブロックアクセスで生じるレイテンシとオーバーヘッドを減少させている。画像全体は３２×３２画素バッファで計算されるので、ｇスプライト画像圧縮は最小オーバーヘッドで行われる。圧縮アーキテクチャの概念的レイアウトを図２０に示す。

変換エンジン６６０(図２０）はモデル及び表示変換、クリッピング、ライティング等を計算し、この情報をタイラー６６２に送信する。タイラーは変換情報を処理すると、テクスチャメモリ６６４からテクスチャデータを読み出す。テクスチャブロックが必要となるので、テクスチャデータは圧縮フォーマットで保存されている。テクスチャブロックはタイラー復元エンジン６６６で復元され、タイラーのオンチップ・テクスチャキャッシュにキャッシュされる。タイラーは、画素データを解析すると、解析されたデータをタイラー圧縮エンジン６６８に送る。ここで解析されたデータは圧縮され、さらに圧縮されたデータはｇスプライトメモリ６７０に保存される。表示制御装置６７２は圧縮されたｇスプライト・データが必要になると、ｇスプライト復元エンジン６７４を使ってｇスプライトメモリ６６４からのｇスプライト・データを復元し、オンチップｇスプライト・キャッシュにデータをキャッシュする。実際のハードウェアでは、テクスチャメモリ664とｇスプライトメモリ６７０は同一である(即ち、圧縮データは様々なエンジンが共有する１つのメモリに保存される）。使用する圧縮方法及び復元方法に互換性があれば、共有する共用

メモリは不要である。またｇスプライト・データはデータベースやその他の画像供給源６７６から取り込んで、テクスチャメモリ６６４及びｇスプライトメモリ６７０に保存することも可能である。

【０２１６】

本発明の一実施例では、損失の多い画素ブロック圧縮及び復元とロスレスの画素ブロック圧縮及び復元の双方をサポートしている。

損失の多い画像圧縮形式には、損失の多い第１ステージとロスレスの第２ステージの２つのステージがある。損失の多い圧縮形式は、赤、緑、青(Ｒ，Ｇ，Ｂ) カラー値から輝度(Ｙ) 及びクロミナンス(Ｕ及びＶ、またＣｒ及びＣｂとも言う) 値への任意のカラー空間変換で始まる。損失の多いステージでは、離散コサイン変換(ＤＣＴ) と所定の周波数成分を減少させる量子化が行われる。

第２ステージは、ハフマンコーディング及びランレングスエンコーディング(ＲＬＥ)から成るロスレスの圧縮形式であるハフマンコーディングの代わりに、算術コーディングなどの他のコーディング方法も使用できる。

損失の多い復元方法では、デコーディング、圧縮データの逆量子化、逆ＤＣＴ、ＹＵＶからＲＧＢへの任意のカラー空間変換が行われる。

ロスレスの圧縮形式では、ＲＧＢからＹＵＶへのロスレスの任意のカラー空間変換、予測ステージ、ロスレスのエンコーディングステージより構成される。このエンコーディングステージは、損失の多い圧縮形式のエントロピーコーディングステージと同じであってもよい。このロスレスの復元方法は、デコーディング、各カラー成分を逆予測するステージ、ＹＵＶからＲＧＢへの任意のカラー空間変換から成り立っている。

損失の多い圧縮／復元

タイラーの圧縮エンジン４１４(図９Ａ～９Ｃ) における損失の多い圧縮方法の１つは、次の４又は５つのステップで行われる。

１．ＲＧＢデータ入力をＹＵＶのような輝度クロミナンス系へ変換する(任意) 。
２．各カラー成分をそれぞれ別個に二次元離散コサイン変換(ＤＣＴ) させる。
３．二次元のＤＣＴ係数をおおよその頻度の増加順に並べる。
４．ＤＣＴ係数を量子化する：均一の除数又は周波数依存性の除数で除算する。
５．求めた係数を、固定コード表を有するハフマンエンコーディングによってエンコードする。

【０２１７】

ロスレスの復元は以下の４又は５つのステップで行われる。

１．圧縮データ入力を、固定コード表を使用するハフマンデコーディングによってデコードする。
２．圧縮データを逆量子化する：圧縮の量子化ステップで使用した均一の乗数又は周波数依存性の乗数で乗算する。
３．線形アレイのデータを適切なＤＣＴ係数の二次元順序に並べる。
４．各カラー成分に対して二次元の逆ＤＣＴをそれぞれ別個に実行する。
５．対応する任意のステップが圧縮処理に含まれる場合、ＹＵＶのような輝度クロミナンス系のカラーをＲＧＢカラーに変換する。

【０２１８】

カラー空間変換

カラー空間変換は、輝度座標Ｙと色差座標Ｕ及びＶに基づいてＲＧＢカラーを輝度クロミナンス系に変換する。この輝度クロミナンス系は標準のカラー空間ではない。カラー座標は輝度の圧縮に必要なビットのほんの一部のみを必要とするため、この系を使用すると圧縮の度合いが向上する。ロスレスの可逆変換は各画素に対して別個に実行され、アルファ値を変えることはない。

【０２１９】

ＲＧＢからＹＵＶ(圧縮の場合)

正数のＲＧＢ値から正数のＹＵＶ値への変換では、次のような変換式を使用する。

Y = (4R + 4G + 4B) / 3 - 512
U = R - G
V = (4B - 2R - 2G) / 3
ＹＵＶからＲＧＢ(復元の場合)
正数のＹＵＶ値から正数のＲＧＢ値への変換では、次のような変換式を使用する。
R = (((Y + 512) - V) / 2 + U + 1) / 2
G = (((Y + 512) - V) / 2 - U + 1) / 2
B = ((Y +512) / 2 + V + 1) / 2
【０２２０】
離散コサイン変換
画像とテクスチャは、３原色の振幅と不透明度の振幅を含む画素である。画素の位置は画像又はテクスチャ・マップ中の空間位置に相当する。この形式の画像やテクスチャは空間ドメインの中にある。画像やテクスチャの場合、離散コサイン変換(ＤＣＴ)でＤＣＴの基底係数を乗じる係数を計算する。画像やテクスチャにＤＣＴを実行すると、画像やテクスチャを同等に表す係数の集合が求まる。この形式の画像やテクスチャは周波数ドメインの中に存在する。
【０２２１】
ＤＣＴでは、８×８画素ブロックのカラーと不透明度の振幅が空間ドメインと周波数ドメインとの間でマッピングされる。周波数ドメインでは、隣接する係数はあまり互いに関連することなく、また圧縮処理で圧縮効率を落とすことなく各関数をそれぞれ別個に処理する。
ＤＣＴでは空間ドメインから周波数ドメインへのマッピングが行われ、逆に逆ＤＣＴでは周波数ドメインから空間ドメインへのマッピングが行われる。ＤＣＴ又は逆ＤＣＴを適切に行なう１つの方法として、１９９０年サンディエゴ、Academic Press Inc. 発行、Rao, K. R.及びP. Yip. 著 "Discrete Cosine Transform"の図Ａ．１．１及びＡ．１．２に開示されている方法がある。
【０２２２】
二次元ＤＣＴにより、各カラー成分を表す周波数ドメインの係数の二次元アレイが生成される。ジグザグの順でその係数が再構成されるので、線形アレイの低い位置で、ＤＣＴの低周波数成分が生じる傾向がある。このジグザグ順において、係数がゼロになる確率は、おおよそ線形アレイでの位置の単調増加関数になる(線形インデックスによる)。このジグザグ順により、知覚量子化及びＬＯＤフィルタリングが簡素化され、ランレングスエンコーディング(ＲＬＥ)も大幅に向上する。
【０２２３】
量子化
量子化を行なうと、係数を正数で除算することでジグザグ順のＤＣＴ係数が有することのできる互いに異なる値の数が減少する。圧縮タイプのパラメータの値によっても異なるが、量子化は均一量子化又は知覚量子化となる。いずれの場合もＤＣ頻度の係数が変わるが(インデックス＝０)、そのままこの係数を送信する。
【０２２４】
量子化処理は、画像又は画像の一部の量子化係数の指定から始まる。この場合、量子化係数は３２×３２画素チャンクに対して指定される。量子化インデックス(Ｑインデックス)はチャンクで使用するための該当する量子化係数(Ｑ係数)を指定する。次の表はＱインデックスとＱ係数との関係を表している。
【０２２５】

【表１】

量子化係数

Ｑインデックス	Ｑ係数	Ｑインデックス	Ｑ係数
0	2	8	32
1	3	9	48
2	4	10	64
3	6	11	96
4	8	12	128
5	12	13	192
6	16	14	256
7	24	15	4096

【０２２６】

各カラー面はチャンクＱインデックスの異なる数値を有している。１５のＱインデックスは４０９６のＱ係数を選択し、これにより量子化及び逆量子化の際にゼロが生成される。量子化処理では、Ｑ係数で各係数を除算し、さらに四捨五入して正数にする。逆量子化処理では、各係数をＱ係数で乗算する。量子化及び逆量子化によってＤＣ頻度成分が変化することはない。

【０２２７】

ブロック量子化係数

Ｑインデックス及びＱ係数はブロック(８×８画素) ごとに変化させることが可能である。あるブロックのＱインデックスは、チャンクのＱインデックスをブロック圧縮タイプに埋め込まれている数値でインクリメントすることで求められる。

【０２２８】

ブロックＱインデックス　＝　チャンクＱインデックス＋(ブロック圧縮タイプ－３)

これはチャンクＱインデックスを１、２、３又は４インクリメントするものである。考えられる最大のＱインデックス数値は１５なので、インクリメントした結果１５を越える数値は１５に設定される。

量子化の種類が知覚量子化である場合、Ｑインデックス及びＱ係数は係数ごとに(アレイインデックスごとに) 変化させることも可能である。

均一量子化の場合は、係数Ｑインデックスはブロックインデックスに等しいので、該当するＱ係数はブロック中の各係数を乗算(逆量子化) 又は除算(量子化) する。

【０２２９】

知覚量子化の場合、係数Ｑインデックスは線形アレイのインデックスの数値(０．．．６３) によって変化する。以下の表はアレイインデックス数値の関数として求めた係数Ｑインデックスを示している。

係数Ｑインデックス	アレイインデックス
ブロックＱインデックス	インデックス＜１２
ブロックＱインデックス＋１	１２≦インデックス＜２８
ブロックＱインデックス＋２	２８≦インデックス＜５２
ブロックＱインデックス＋３	５２≦インデックス

エントロピーコーディング

【０２３０】

ハフマン／ＲＬＥコーディングは量子化されたＤＣＴ係数を以下の方法で処理する。

１．(ＤＣＴ係数は互いに関連していないため) ゼロ以外の係数を存在しうる最低ビット

で別個にエンコードする。
２．(ジグザグ順であるため) 特に線形アレイの最後にあるゼロの数値で連続する種類の係数を最適にエンコードする。
ハフマン／ＲＬＥコーディング処理を適切に行なう方法としては、既知のＪＰＥＧ静止画像圧縮標準のAC係数で使用されるハフマン／ＲＬＥコーディング処理がある。
【０２３１】
ブロックにランダムアクセスするために、この方法ではDCの周波数係数(インデックス＝０）をエンコードしないが、数値を変えずにこれを送信する。
アルゴリズムは、以下を表す一連の可変長コードを計算する。
１．次のゼロ以外の係数の前の一続きのゼロの１から１５までの長さ。
２．次のゼロ以外の係数の符号と仮数を指定するのに必要な追加ビット数。
コード語の後にゼロ以外の係数の符号と仮数が書き込まれる。ある１つの指定コードは、ブロックの残りの係数が全てゼロであることを表す。
【０２３２】
エンコーディング
全ブロックをエンコードする場合、ＩＳＯ国際標準１０９１８の付録Ｋ、セクションＫ．３．２に記載されている代表的なＡＣ係数のハフマン表を使用する。これには、輝度(Ｙ）ＡＣ係数の表Ｋ．５とクロミナンス(Ｕ及びＶ)ＡＣ係数の表Ｋ．６が示されている。
【０２３３】
デコーディング
全ブロックをデコードする場合、エンコード処理として同一の一定表を使用する。従って、データを書き込んだハフマン表を保存する必要はなく、移動させる必要もない。
【０２３４】
ロスレスの圧縮／復元
タイラーの圧縮エンジン４１４では、次の２つのステージでロスレスの圧縮が行われる。
１．入力されたＲＧＢデータからＹＵＶのような輝度クロミナンス系への変換(任意)。
２．各カラー成分に対して微分予測計算を行なう。固定コード表を使用するハフマンエンコーディングにより、求めた係数をエンコードする。
タイラーの復元エンジン４０４，４５０及びｇスプライト・エンジンでは、次のような２又は３つのステージでロスレスの復元が行われる。
１．固定コード表を使用するハフマンデコーディングにより、入力される圧縮データをデコードする。
２．各カラー成分に対して逆の微分予測(再構成) 計算を行なう。
３．圧縮処理にＹＵＶのような輝度クロミナンス系のカラーをＲＧＢカラーに変換する任意のステップが含まれる場合、この変換処理を行なう。
【０２３５】
カラー空間変換
カラー空間変換は、ＲＧＢカラーを輝度座標Ｙとカラー座標U及びVに基づく輝度クロミナンス系に逆変換する。これはＹＵＶ系以上に圧縮の度合いを上げることのできる他と異なるカラー空間であるハフマン／ＲＬＥエンコーダへの入力数が少なく、さらなる圧縮が可能となるためである。各画素はそれぞれ別個にカラー空間変換され、またこの変換によってアルファの値が変化することはない。
【０２３６】
ＲＧＢからＹＵＶ(圧縮の場合)
正数のＲＧＢ値から正数のＹＵＶ値へ変換する場合、次のような変換式を使用する。
Ｙ = Ｇ
Ｕ = Ｒ - Ｇ
Ｖ = Ｂ - Ｇ
【０２３７】

ＹＵＶ からＲＧＢ(復元の場合)

正数のＹＵＶ値から正数のＲＧＢ値へ変換する場合、次のような変換式を使用する。

R = Y + U

G = Y

B = Y + V

【０２３８】

カラー空間変換の実行中、アルファ情報は変化しない。

カラー空間変換を回避することも可能である。ｇスプライト制御データ構造でフラグを用いて、カラー変換を回避することをデコンプレッサに知らせる。

カラー空間変換後に予測が行われる。予測は、大部分の元画像、特に空き空間と水平及び垂直線を多く含む画像のエントロピーを減少させるロスレスの可逆処理である。

【０２３９】

圧縮の予測処理及び復元の逆予測処理においては、以下のような値が使用される。

１．ｐ(ｘ，ｙ)は、コンプレッサに入力され、また復元エンジンから出力される画素値である。

２．ｄ(ｘ，ｙ)は、圧縮エンジンの次段のコーダへ入力され、また復元エンジンのコーダの反対側から出力される差分値である。

【０２４０】

予測は次のように計算される。

x = 0, y = 0のとき、d(x,y) = p(x,y)

x = 0, y > 0のとき、d(x,y) = p(x,y) - p(x,y-1)

x > 0 のとき、d(x,y) = p(x,y) - p(x-1,y)

復元エンジンでの逆予測は次のように計算される。

x = 0, y = 0のとき、p(x,y) = d(x,y)

x = 0, y > 0のとき、p(x,y) = p(x,y-1) + d(x,y)

x > 0 のとき、p(x,y) = p(x-1,y) + d(x,y)

ハフマン／ＲＬＥコーディング及びデコーディングは、本実施例の損失の多い圧縮／復元形式で行われるものと同一である。

【０２４１】

上記の圧縮方法では、８×８画素の独立ブロックの画像を圧縮する。従って上記のチャンキング・アーキテクチャでは、圧縮された各３２×３２画素チャンクはこのような１６個のブロックで構成される。３２×３２画素チャンクの圧縮を容易にするために、アンチエイリアシング・エンジン４１２は画素データを8×８画素ブロックに変換する。８×８画素ブロックは、第１のバッファで保存を行い、また第２のバッファが圧縮されるように、８×８画素ブロックをバッファする。

【０２４２】

制御及びパラメータ

上記に述べたように、タイラー(図９Ａ～９Ｃ)は一度にｇスプライトを１つのチャンクにレンダリングする。このようなチャンクは画素ブロックで構成されている(この場合、１６個の８×８画素ブロック)。テクスチャ・マッピング、陰影付け、その他のマルチパス・レンダリング動作の場合、タイラーはｇスプライト又はテクスチャのブロックをメモリから取り込む。フレームを構成するために、ｇスプライト・エンジン(図１２Ａ～Ｂ)はｇスプライトを取り込み、画素を画面空間に変換し、合成バッファで画素を合成する。

【０２４３】

ｇスプライト、チャンク、ブロックの処理を制御する制御パラメータは複数存在する。ｇスプライト表示リストは表示画像を構成するｇスプライトのリストを保存している。以下に説明するように、ｇスプライト・ヘッダブロックは、ｇスプライトの幅や高さを含むｇスプライトの属性、及び画面空間において平行四辺形で規定されているアフィン変換式を多数保存している。ｇスプライト・ヘッダブロックも、その関連するチャンクのリスト

を有している。一実施例では、このリストは、チャンク制御ブロック用のポインタ又はハンドルの形式とする。

【０２４４】

チャンク制御ブロックは、チャンク毎及びブロック毎のパラメータを有している。チャンク毎のパラメータは、ＹＵＶカラー変換バイパスと、デフォルトＱ係数と、知覚量子化フラグと、画素フォーマットと、画素データが線形メモリのメモリ割り当てユニット(ＭＡＵ) で管理されるメモリの中に存在するか否かを表す情報とで構成されている。ＭＡＵはチャンクメモリを割り当てるのに使用される１つの共用メモリである。ＭＡＵが管理するメモリにはＭＡＵのリストが保存されており、各ＭＡＵは次のＭＡＵを示すポインタを有している。例えばある具体例においては、チャンク制御ブロックは各ｇスプライト毎に連続するMAUの中に保存される。

【０２４５】

ブロック毎のパラメータは、圧縮タイプと、ブロックが保存されるＭＡＵの数と、ブロックの画素データの第１番目のバイトを示すブロックポインタとで構成される。具体的なブロックのフォーマットは、３２ビット画素(８ビットはＲＧＢ及びアルファ用) をエンコードする８×８ｘ４画素アレイである。

上記の制御パラメータを使ってｇスプライトの画素を有する座標(Ｘ，Ｙ) を検索するステップは以下の通りである。

１）３２でＹ及びＸを除算し、チャンクの列と行をそれぞれ得る。

２）(チャンク列) ＊(チャンクの中のスプライトの幅) ＋チャンク行を計算して、チャンク数を得る。

３）(チャンク数) ＊（チャンクヘッダブロックのサイズ） を計算して、チャンク制御ブロックのオフセットを求める。

４）(Ｙ＜４：３＞＊４＋Ｘ＜４：３＞)＊３を計算して、チャンク制御内のブロックのオフセットを求める。

５)ブロックポインタを復元したキャッシュロジックに送り、ブロックを受信する。

６)（Ｙ＜２：０＞＊８）＋Ｘ＜２：０＞を計算して、ブロック内の画素のオフセットを求める。

【０２４６】

ここにおいて、チャンクのオフセットはチャンクを選択するのに使用される。また、ブロックオフセットはブロックポインタの選択で用いられる。

ブロックポインタは画素を含むブロックを選択し、また画素のオフセットは画素を選択する。

画素データの圧縮ブロックの中にある所定の画素に対応するブロックにアクセスするために、タイラー及びｇスプライト・エンジンのキャッシュ制御が以下のステップを行なう。

１）チャンク制御ブロックにあるブロックポインタ値を検索し、ＭＡＵのサイズで除算することにより、ＭＡＵアドレスを求める。

２）そのブロックに対応するチャンク制御ブロックに割り当てられているＭＡＵの数を検索する。

３）チャンク制御ブロックの次のブロックポインタドレスを検索する。

４）割り当てられているＭＡＵの数　＊　ＭＡＵサイズ　＋ ((ブロックポインタ) 変更ＭＡＵサイズ) ＋(次のブロックポインタ) 変更(ＭＡＵサイズ) の２の補数

５）ブロックアドレス及び圧縮ブロックの長さを圧縮キャッシュのロジックに送る。

【０２４７】

圧縮キャッシュは第１のＭＡＵを読み出し、転送長さが十分でなければＭＡＵの中のポインタを使用して次のＭＡＵの先頭にアクセスする。この処理は転送長さが十分になるまで続けられる。

ＭＩＰマップテクスチャ動作をサポートするために、タイラーは他のインデクシングレベルをサポートしている。ＭＩＰマップレベルのインデクシングを行なう１つの方法では

、次のようなステップを行なう。
１）所定のスプライトに対して、以下の計算を行ってＭＩＰチャンクレベルオフセットの表を作成する。
ＭＩＰチャンクオフセット［０］＝　０　＼＼ディテールのレベル０のオフセット
ＭＩＰマップの各レベルの場合：
ＭＩＰチャンクオフセット［レベル＋１］＝スプライト幅／(２＾レベル) ＊スプライト高さ／(２＾レベル) ＋ＭＩＰチャンクオフセット［レベル］
２）ＬＯＤパラメータを使用してＭＩＰチャンクオフセットを求める。
このとき、ＭＩＰチャンクオフセット、スプライト幅／(２＾レベル) 、スプライト高さ／(２＾レベル) を使用して、現在のｇスプライトの選択したディテールのレベル内にある所望のチャンクを求めることもできる。
ｇスプライト
上記にｇスプライトの概念を説明した。簡単にもう一度説明すると、視体積中の１つ以上のオブジェクトはｇスプライトに割り当てられる。ｇスプライトは別個にレンダリングされるので、これによりｇスプライトを種々の解像度でレンダリングさせ、また可変の速度で更新させることができる。レンダリング・オーバーヘッドを低減させるために、システムはオブジェクトを再レンダリングせずに、ｇスプライトをアフィン変換させることにより、オブジェクトの動きを近似的に作り出すこともできる。画面を構成するｇスプライトを表示させるために、システムは画面の中でオブジェクトを表すｇスプライトを合成する。次に、上記及び他の特徴について詳しく説明する。
上記に述べたように、本システムは、幾何形状をｇスプライトに割り当てることにより始まる。ｇスプライトは物理的出力装置の座標で示される二次元領域である。以下に述べる例では、ｇスプライトの形は矩形であるが、他の形を使用することも可能である。ｇスプライトはｇスプライト・エンジンでアフィン変換させることができる(即ち、スケール、回転、反転、及び／又は切り取りが可能である－いずれの変換も２ｘ２行列を使用した変換と移動である) 。二次元変換の一適用例としては、三次元移動のシミュレーションがある。ｇスプライトの例として、同一のｇスプライトの画像が様々に変換されて複数回画面に表示させることが可能である。この例はｇスプライト画像の矩形のサブセットや画像全体に適用できる。また、カラー成分をベースにして適用させることも可能である。例えば、アルファ値をあるｇスプライトから得て、カラー値を別のｇスプライトから得ることもできる。
一般に、画像前処理装置１０４は１つのオブジェクトを単一のｇスプライトに割り当てるが、複数のオブジェクトを１つのｇスプライトに割り当てることも可能である。画像前処理装置１０４は互いに影響を及ぼすオブジェクトや他と関連性を持たないオブジェクトを単一のｇスプライトの中で組み合わせる。また、メモリと処理制限に基づいてオブジェクトを統合する。例えば、画像処理装置は、それぞれ独立しているが重なり合っているｇスプライトを出力装置のリフレッシュ速度が要する時間内で合成することはできない。この場合、システムはこのような重なり合っているオブジェクトを単一のｇスプライトに統合する。
【０２４８】
オブジェクトをｇスプライトに割り当てた後、画像処理装置はフレームのｇスプライトをレンダリングする。別個にオブジェクトをレンダリングすると、システムはレンダリング・オーバーヘッドを減少させることができる。これは、各フレームの画面で各オブジェクトを再レンダリングする必要がないためである。この特徴についてさらに詳細に説明する。
【０２４９】
オブジェクトを画面に表示させるために、画像処理装置１０６は画面のオブジェクトを含むｇスプライトを合成する。合成とは、ｇスプライトの層からのカラーデータを組み合わせる処理のことをいう。半透明性をサポートするために、画像処理装置１０６は合成して表示を行なう際に変換したｇスプライト画素のアルファ値も考慮に入れる。

【０２５０】
図２１Ａ及び２１Ｂは、一実施例においてのｇスプライトの処理方法を示すフローチャートである。図に示す実施例では、ｇスプライトの処理は２フレーム期間にわたって行われる。最初のフレーム期間において、画面のオブジェクトはｇスプライトに割り当てられレンダリングされる。また次のフレーム期間において、ｇスプライトは変換され、さらに合成される。
まず、画像前処理装置１０４は潜在的に可視のオブジェクトを決定する。図２１Ａはこの処理を一連のステップとして示している。フレーム毎に、画像処理装置１０６はオブジェクトのリストを移動させ、画面、即ちビュー空間において潜在的に可視のオブジェクトをトラバースすることで、潜在的に可視のオブジェクトを決定する(６９６，６９８)。
【０２５１】
次に、画像前処理装置１０４はｇスプライトの割り当て、再割り当て、割り当て解除を行なう。ｇスプライトの割り当てとは、一般的にシステムでｇスプライトを表わすためにデータ構造を作成することをいう。オブジェクトが潜在的に可視のオブジェクトでなく(７００)、システムがそのオブジェクトに対してｇスプライトを割り当てていない場合は(７０２)、他の処理を行なう必要はない。オブジェクトが潜在的に可視のオブジェクトであり(７００)、システムがそのオブジェクトに対してすでにｇスプライトを割り当てている場合は(７０２)、画像前処理装置１０４は、そのオブジェクトに対するｇスプライトの割り当てを解除する(７０４)。
【０２５２】
画像前処理装置１０４は、システムがまだｇスプライトを割り当てていない潜在的に可視のオブジェクトに対して新しいｇスプライト・データ構造を割り当てる(７０６，７０８)。この場合、画像前処理装置１０４はｇスプライト・データ構造を作成し、レンダリングを行なうために、そのオブジェクトに対応する画像データをキューする（７１０)。レンダリングを行なうために「キューする」というのは、三次元レンダリング用のオブジェクトのリストに追加することを表している(７１０)。また画像前処理装置１０４はｇスプライトのアフィン変換式を計算する(７１４)。本実施例では、アフィン変換式には２つの目的がある。第１に、アフィン変換式は、画面上での動きに対応するオブジェクトの動きを近似的に作り出すことができる。第２に、アフィン変換式は、ｇスプライトをｇスプライトの空間から出力装置座標に変換することができる。ｇスプライト空間とは、オブジェクトをチャンクに下位分割させるのに使用する座標系を意味している。オブジェクトをチャンクにサブ分割する際に使用する座標系は、二次元空間に変換されたオブジェクトを効率よくチャンク領域で扱うことができるように最適化することが可能である。
【０２５３】
オブジェクトが潜在的に可視のオブジェクトであり(７００)、本システムがそのオブジェクトに対してすでにｇスプライトを割り当てている場合は(７０６)、図に示す画像前処理装置１０４はアフィン変換の計算を行なう(７１４)。以下に詳細に説明するが、アフィン変換を利用してオブジェクトの動きを擬似的に作り出すことが可能である。画像前処理装置１０４はこの近似的に作り出した動きの精度を評価し、歪みが多い場合は(７１６)、画像前処理装置１０４がそのオブジェクトに対してｇスプライトの再割り当てを行なう(７０８)。この場合、次に画像前処理装置１０４は、レンダリング用のｇスプライトへとレンダリングされるべき幾何形状をキューして(即ち、三次元リストに置く)（７１０)、ｇスプライトを表示リストに追加する(７１８)。
【０２５４】
しかし、アフィン変換式を用いてオブジェクトの動きを正確に擬似的に作り出すことができる（歪みが所定の許容範囲内にある） 場合は(７１６)、オブジェクトをレンダリングし直す必要はなく、画像前処理装置１０４はそのオブジェクトに対応するｇスプライトを表示リストに入れる(７１８)。
【０２５５】
次のフレーム期間に、画像処理装置１０６は表示画像を生成する。点線で分けられてい

るステップ(７１８)及び(７２０)はフレーム期間を示している。画像処理装置１０６は表示リストを移動させて、リストのｇスプライトを物理的出力装置の座標に変換する(７２０)。一般に、出力装置座標への変換には、ワープや回転、拡大・縮小を施したｇスプライトから出力装置の画素位置へと画素データを走査することを含む。次に画像処理装置１０６はこの変換又は「走査」したｇスプライト・データを合成する(７２２)。最後に、画像処理装置１０６は画素データをアナログ値に変換し、画像を表示する。

【０２５６】

図５Ａ及び５Ｂはチャンキング・アーキテクチャで幾何形状をレンダリングする処理を示すフローチャートである。重要なことであるが、上記に述べたｇスプライトの概念はチャンキング・アーキテクチャにのみに限定されるものではない。図５Ａ及びそれに関する上記の説明では、画像前処理装置１０４が画面内の幾何形状からｇスプライト構成を確定する方法について詳しく述べている。ステップ(２４０－２４４)及びそれに関する説明を参照されたい。特に、オブジェクトを、単一のｇスプライト、又は必要に応じて幾つかのｇスプライトに統合及びレンダリングすることが可能である。例えば、タイラー、ｇスプライト・エンジン、合成バッファに必要とされるフレーム・リフレッシュレートで、あるフレームのｇスプライトに対して、現在のオブジェクトの割り当てを処理することができない場合、データをＤＳＰ１７６又はプロセッサ１３２に再び送信し、オブジェクトを統合してｇスプライトに複数のオブジェクトをレンダリングすることが可能である。

【０２５７】

図６は、一実施例におけるｇスプライトの処理に関して更なる知見を与えるものである。図６に示されるとともに上述した如く、画像前処理装置１０４は、ｇスプライトの深度順も決定する(２８０)。

【０２５８】

画像前処理装置１０４がｇスプライトを割り当てるとき、それはｇスプライトを表すデータ構造を生成する。このｇスプライト・データ構造は、ｇスプライトの種々の属性を記憶するとともに関連する画像データがメモリの何処に在るかを追跡し続ける為のヘッダを含んでいる。このデータ構造は、ｇスプライトのサイズを記憶し、ｇスプライトのエッジに対するエッジ方程式を表すための、及び、２Ｄ 変換データ及び他の画像の属性を保持するための種々のフィールドを含んでいる。

【０２５９】

ビュー空間に対するｇスプライトの配置形態を決定した後、画像処理装置はどのｇスプライトをレンダリングするかを決定する。場面における全てのオブジェクトをレンダリングする代わりに、システムは他のフレームからレンダリングされたｇスプライトを再使用することができる。また、フレーム毎のオブジェクトの位置の変化は、レンダリングされたｇスプライトのアフィン変換を行なうことにより近似される。図６に示される如く、画像処理装置はｇスプライトをループ(２８２－２８６) して、ｇスプライト変換を計算する(２８４) 。尚、以下においては、ｇスプライトの更新及びｇスプライトのワープに関して詳述する。

【０２６０】

本画像処理システムは、レンダリングされてオブジェクトを表す２Ｄ ｇスプライトに関してアフィン変換を行なうことにより３Ｄ オブジェクトの動きを近似することができる。尚、本明細書においては、レンダリングされた画像に関するアフィン変換を“ワープ(warping) ”と称するとともに、この処理の結果として得られるｇスプライトを“ワープされたｇスプライト(warped gsprite) ”と称する。ひとつの実施形態においては、オブジェクトの３Ｄ レンダリングをシミュレートする処理は、１)特徴点の幾何的な動きを近似するアフィン変換行列を算出するステップと、２)ステップ1 における近似の精度を測定するステップと、３)精度が十分であれば、時間ｔ$_0$におけるｇスプライトのアフィン変換を行なうことにより後の時点t におけるそれの位置を近似するステップとを含んでいる。

【０２６１】

図２２は、３Ｄ での動きをシミュレートするアフィン変換を行なう処理を示すフローチャートである。詳細に述べると、図２２は最初のステップ７４４で“特徴点選択”を示している。以下の検討内容から明らかとなる如く、画像処理の間に特徴点は選択されず、幾何モデルのオーサー(author)により指定されるのが通常である。

【０２６２】

オブジェクトの動きをシミュレートするために使用されたアフィン変換は、特徴点を用いて計算される。特徴点は、オブジェクトの位置又は経時的に変化する他の重要な画像特性を表すべく選択された点である。此処では、３Ｄ モデルのワールド座標系と画面空間に変換されたモデルの画面座標系とにおける特徴点に言及することから、これらの点を記述すべく使用する語句を明確にすることは有用であろう。此処では、画面空間における特徴点を“ビューイング特徴点(viewing characteristic points) ”と称し、且つ、ワールド座標系における特徴点を“モデル特徴点(modeling characteristic points)”と称する。

【０２６３】

オブジェクトの全体を考慮するのではなく特徴点の内で代表的なものを選択することにより、アフィン変換の計算は相当に簡略化される。オブジェクトの３Ｄ での動きを正確に近似する為に必要とされる特徴点の個数は、モデルに依存して変化する。オブジェクトが剛体であれば、オブジェクト全体を囲む境界ボックスから特徴点を選択することができる。また、境界ボックスを規定する点を同一の変換法により変換する場合、境界ボックスのその点はオブジェクト幾何形状の変換に従う。

【０２６４】

また、より複雑な動きを行なうオブジェクトに対しては、正確な近似を行なう上では更に多くの特徴点が必要とされよう。例えば、オブジェクトは、個々の位置を近似する境界ボックスをそれぞれ有する多数の剛体にサブ分割することが可能である。オブジェクトが、独立した移動変換(moving transformations)を有する階層構造の剛体のオブジェクトからなるものであれば、移動するサブオブジェクトの境界ボックスを結合したものから特徴点を導出することができる。

【０２６５】

別の代替例としては、モデルのオーサーがモデルの特徴点を指定することも可能である。この様にすれば、オブジェクトの３Ｄ 移動を近似すべく使用された特徴点をオーサーが詳細に指定することが可能となる。以下に更に記述する様に、アフィン変換の精度は多くの距離(metrics) のいずれかに依って確認することが出来る。オーサーが特徴点を指定するようにすることで、アフィン変換の精度を評価すべく使用される1つ以上の距離に対して適切な点をオーサーが指定できるようになる。

【０２６６】

特徴点の組が与えられたとき、アフィン変換を計算することにより時間t_0から時間ｔへのｇスプライトの位置変化を近似することができる。このステップは図２２のステップ(７４６) に示されている。

アフィン変換は、時間t_0及び時間ｔ におけるビューイング特徴点から計算される。特徴点が選択された方法に依存し乍ら、モデル特徴点はオブジェクト上、又はその境界ボックス上の点を表す。これらのモデル特徴点の位置は、モデル変換(modeling transform)に係る時間によって変化する。また、ビューイング特徴点を見出す為には、モデル特徴点にビューイング変換が乗ぜられる。更に、以下の説明は、２Ｄのｇスプライトを変換する上で用いられるアフィン変換行列を計算する処理を明確化する上で有用である。

アフィン変換行列のフォーマットは次の如きものである：

【０２６７】

【数１】

$$S=\begin{bmatrix} a & b & p_x \\ c & d & p_y \end{bmatrix}$$

【０２６８】

近似の精度を確認するためのひとつの距離は、位置距離(position metric) である。位置距離とは、アフィン変換行列を乗じた、時間ｔ における特徴点と時間ｔ$_0$における特徴点の位置との間の差を指している。位置距離の為の一般式は次の如きものである：

【０２６９】

【数２】

$$\sum_i \left\| \bar{x}^i(t) - S(t)\bar{x}^i(t_0) \right\|^2$$

【０２７０】

位置距離の場合、画面空間内の特徴点の位置が最も適切である。なぜなら、画面上の位置の差は、変換したｇスプライトが対応３Ｄ モデルの動きを如何に正確に近似しているかを表すからである。但し、他の距離に対しては、近似の精度をモデル特徴点に関して計算することも可能である。位置距離の例として、画面空間の点を直接的に考察する。ここで、Ｖ(ｔ)をビューイング変換とするとともにＴ(ｔ)をモデル変換とし、

【０２７１】

【数３】

$$\bar{x}^i(t) = V(t)T(t)x^i(t)$$

を画面空間の点とする。アフィン変換行列を計算するには、標準的な最小２乗法による解法を使用することが可能である。以下の線形システム、

【０２７２】

【数４】

$$[\bar{x}^i(t_0)\ 1]S(t) = \bar{x}^i(t)$$

を解けば、標準的な最小２乗法による解法は位置距離を最小化する解を与える。

【０２７３】

３個の特徴点が在る場合は、アフィン変換行列を直接的に解くことが可能である。例えば、境界ボックスの軸心の３個の点を使用するのであれば、時間に依存する以下のアフィン変換行列に対して解は閉じた形の表現となる：

【０２７４】

【数５】

$$\begin{bmatrix} x^0 & y^0 & 1 \\ x^1 & y^1 & 1 \\ x^2 & y^2 & 1 \end{bmatrix}_{t_0} \begin{bmatrix} a & c \\ b & d \\ p_x & p_y \end{bmatrix}_t = \begin{bmatrix} x^0 & y^0 \\ x^1 & y^1 \\ x^2 & y^2 \end{bmatrix}_t$$

$$S(t) = [X(t_0) \quad 1]^{-1} X(t)$$

$$[X(t_0) \ 1]^{-1} = \frac{1}{D} \begin{bmatrix} y^1 - y^2 & y^2 - y^0 & y^0 - y^1 \\ x^2 - x^1 & x^0 - x^2 & x^1 - x^0 \\ x^1 y^2 - x^2 y^1 & x^2 y^0 - x^0 y^2 & x^0 y^1 - x^1 y^0 \end{bmatrix}$$

ここに、$D = x^1y^2 - x^2y^1 + x^2y^0 - x^0y^2 + x^0y^1 - x^1y^0$

【０２７５】

一般的な場合、アフィン変換行列を解く上では正規方程式(normal equations)、又は特異値分解(singular value decomposition)などの最小２乗法を用いることが可能である。一般化した場合は以下の様になる：

【０２７６】

【数６】

$$\begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ . & . \\ . & . \end{bmatrix} \begin{bmatrix} a & c \\ b & d \\ t_x & t_y \end{bmatrix} = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ . & . \\ . & . \end{bmatrix}$$

【０２７７】

アフィン変換行列を解く為には、Ｎｘ３行列の擬逆行列(pseudo inverse) を計算せねばならない。また、任意の個数の特徴点に対しては、最小２乗法を用いて擬逆行列を解くことになる。ひとつの実施例においては、正規方程式の方法を用いる。

【０２７８】

【数７】

また、$\tilde{X}$を時間 t_0 における特徴点の転置行列 (transposed matrix) とし、

Xを時間 t における特徴点の転置行列とすれば、次式のように表される。

$$[\tilde{X} \ 1] S = X$$

【０２７９】

【数８】

$$\begin{bmatrix} \tilde{x}^0 & \tilde{y}^0 & 1 \\ \tilde{x}^1 & \tilde{y}^1 & 1 \\ \vdots & \vdots & \vdots \\ \tilde{x}^{n-1} & \tilde{y}^{n-1} & 1 \end{bmatrix} \begin{bmatrix} a & c \\ b & d \\ p_x & p_y \end{bmatrix} = \begin{bmatrix} x^0 & y^0 \\ x^1 & y^1 \\ \vdots & \vdots \\ x^{n-1} & y^{n-1} \end{bmatrix}$$

【０２８０】
正規方程式の方法で解くためには、式の両辺に適合行列(fitting matrix)の転置行列を乗じ、次に、得られた正方行列を反転する。正規方程式の典型的な欠点は、得られた行列が特異になり、又は丸め誤差により不安定になり易い、ということである。特徴点が縮退すれば、行列は特異となる。行列の特定の形態においては、各項を正規化することにより丸め誤差を調整することが可能である。
【０２８１】
【数９】

$$[\tilde{X}\ 1]^T\ [\tilde{X}\ 1]\ S = [\tilde{X}\ 1]^T\ X$$

$$\begin{bmatrix} \sum_i \tilde{x}^i\tilde{x}^i & \sum_i \tilde{x}^i\tilde{y}^i & \sum_i \tilde{x}^i \\ \sum_i \tilde{x}^i\tilde{y}^i & \sum_i \tilde{y}^i\tilde{y}^i & \sum_i \tilde{y}^i \\ \sum_i \tilde{x}^i & \sum_i \tilde{y}^i & 1 \end{bmatrix} \begin{bmatrix} a & c \\ b & d \\ p_x & p_y \end{bmatrix} = \begin{bmatrix} \sum_i \tilde{x}^i x^i & \sum_i \tilde{x}^i y^i \\ \sum_i \tilde{y}^i x^i & \sum_i \tilde{y}^i y^i \\ \sum_i x^i & \sum_i y^i \end{bmatrix}$$

【０２８２】
得られた行列には丁度５個の項がある。３ｘ３ 行列は次に反転されて、アフィン変換式が得られる。代わりに、ｘ 座標の項の和とｙ 座標の項の和は特徴点の図心に対応することから、座標形を変更して図心を０，０ に平行移動することにより、これらの項は消去することができる。得られる行列は２ｘ２であり、容易に反転される。
【０２８３】
アフィン変換行列を計算した後、近似の精度をひとつ以上の距離を用いてチェックする。図２２の判定ステップ(７４８) は、ひとつ以上の距離をチェックするステップを示すとともに、ロジックが距離に基づいて如何にして分岐するかを概略的に示している。上述の如く、位置距離は、アフィン変換の精度を如何にチェックするかのひとつの例である。アフィン変換が位置距離を満足するか否かを測定すべく、計算されたアフィン変換を用いて変換された時間ｔ$_0$におけるビューイング特徴点を、時間ｔ におけるビューイング特徴点と比較する。
【０２８４】
別の試みは、３Ｄ モデルの内部回転を距離として使用することである。この場合、計算されたアフィン変換を用いて変換された時間ｔ$_0$におけるモデル特徴点は、時間ｔにおけるモデル特徴点と比較される。
更に別の試みは、ライティング距離(lighting metric) を使用することである。内部回転に対する距離と同様に、モデル特徴点を用いて近似の精度をチェックするのである。
【０２８５】
上述の距離に加え、他の種々の代替物が在る。これらの距離を計算する為に、特徴点に沿って適切な特性データが保持され得る。また、所望の精度に依り、１つの距離又は各種の距離を組合せて使用することが可能である。
変換されたｇスプライトを表す特徴点が十分に正確であれば、変換されたｇスプライトを、再レンダリングされたｇスプライトの代わりに使用することが出来る。２Ｄ 変換を計算する為に、時間ｔ$_0$に対するｇスプライトにアフィン変換行列を乗ずる(７５０)。ｇスプライトのレンダリングと対照的に、この計算では処理時間が極めて短い。従って、２Ｄ 変換による３Ｄ移動のシミュレートすることは、画像をレンダリングするに必要な処理量を相当に減少させるものである。
【０２８６】
近似の精度に基づき、システムは画像データのフレームに対し、そのレンダリング能力の範囲内に納めるに必要な程度までレンダリングの総量を減少することができる。この概念を概略的に示すべく、図２２は、２Ｄ 変換が十分に正確でなければｇスプライトが再

レンダリングされることを示している。しかし乍ら、以下に更に詳述する如く、距離に基づいてｇスプライトを受け入れ、又は排除することは必ずしも望ましいことではない。正確に述べれば、或る場面における多くのｇスプライトに対する近似が如何に正確かを決定してから、可能な限り多くのｇスプライトを再レンダリングすることが有用であることが多いからである。

【０２８７】

ｇスプライトのカラーワープ(color warping) ：

更なる最適化として、レンダリングシステムはフレーム毎のライティング変化をサンプリングするとともにｇスプライトのカラー値を改変してこれらの変化を近似することが出来る。この試みは、１)フレーム間のライティング変化をサンプリングし、２)ｇスプライトのカラー値を改変してライティング変化を近似する（即ち、カラーワープを計算する）方法を決定し、かつ、３)十分に正確であればｇスプライトに関してカラーワープを行ってライティング変化を近似する、という基本的な３つのステップを含んでいる。ライティング式の評価の後に、ライティングが所定量以上変化したことを画像前処理装置１０４が決定したのであれば、それはタイラー(tiler) に対してオブジェクトの再レンダリングを指示する。

【０２８８】

第１ステップにおいて、レンダリングシステムはｇスプライトと組合わされたオブジェクトに対するライティング変化をサンプリングする。それは、オブジェクトがｇスプライトにレンダリングされた最初のフレームと、レンダリングシステムがｇスプライトのカラーワープを行ってカラー変化を近似せんとする次フレームとの間のライティング変化をサンプリングする。ここで、ライティング変化をサンプリングするためのひとつの手法は、最初のフレームと次フレームとに対する法線による特徴点におけるライティング式をサンプリングし、これらのフレームの各々におけるサンプリング結果を比較することである。好適には、特徴点をオブジェクト上に分布させ、ｇスプライトに跨がるライティング変化を正確にサンプリングせねばならない。特徴点の詳細な個数及び位置は、変更可能であり、一般的には、モデル特有のものとなる。

【０２８９】

ライティング式の一例を以下に示す。

【０２９０】

【数１０】

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + f_{att}\,I_{p\lambda}[k_dO_{d\lambda}(N \bullet L) + k_s\,O_{s\lambda}(R \bullet V)^n]$$

【０２９１】

ここに、

$I_{a\lambda}$は周囲の明るさ、

k_aは周囲の反射係数、

$O_{d\lambda}$はオブジェクトの拡散カラー、

f_{att}は、光エネルギが光源から離間するにつれて如何に減少するかを記述する、光源の減衰率、

$I_{p\lambda}$は点光源からの光、

k_dは、物質毎に変化する、０と１の間の定数である拡散反射係数、

$O_{s\lambda}$は、オブジェクトの反射カラー、

k_sは、０から１までの物質の鏡面反射係数、

(N・ L)は、面法線N と光源L の方向との間の内積、

(R・ V)は、反射方向R と視点V との間の内積、

上添字n は、物質の鏡面反射「べき指数」であり、通常は、１から数百まで変化し、

λは、この下添字を有する項は波長に依存することを示している。ライティング式を簡略化する為のひとつの仮定は、オブジェクトに対する光の相互作用をＲＧＢ カラーモデルが十分に表現し得る、ということである。この仮定に基づき、ライティングモデルをＲ

、Ｇ 及びＢ のカラー成分の各々に適用することが可能となる。

【０２９２】

上記のライティング式は、オブジェクトの面上の点におけるライティングを計算する方法のひとつを例示しているに過ぎない。ライティング式は、例えば、光減衰率、又は鏡面反射を無視することにより、簡略化することができる。３Ｄ グラフィックレンダリングの分野においては、グラフィックオブジェクトの面上のライティングを表現すべく使用される他の習用のライティング式が種々存在する。従って、グラフィックオブジェクトと関連付けされた特徴点におけるライティングをサンプリングする上では、多数の異なるライティング式の内の任意のものを使用することができる。一般的に、画像前処理装置１０４はライティング式を計算し、（通常、ＲＧＢ 成分の各々に対する）結果として生じるライティング値Iがフレーム毎に如何に変化するかを決定する。

【０２９３】

フレーム毎のライティング変化を評価する為に、画像前処理装置１０４は最初のフレーム及び次フレームにおける特徴点に対するライティング式を計算するが、これは、特徴点における直交面、各フレームに対する光源方向と、通常、特定のライティング式に伴う他のデータとを用いて計算される。

【０２９４】

システムは、ｇスプライトにより表されるオブジェクト上の特徴点、又はオブジェクトの境界ボリューム上の特徴点におけるライティング変化をサンプリングすることが出来る。ライティング変化をサンプリングする為のひとつの試みは、オブジェクトの境界ボリュームの面上のライティング変化をサンプリングすることである。例えば、本システムは、オブジェクト、又はオブジェクトの一部の境界球上の法線におけるライティング変化をサンプリングすることが可能である。境界球は、画像前処理装置１０４に対し、オブジェクトの“空間”内で移動しつつある局部的な光源により生じる重要な変動の追跡を許容する。画像前処理装置１０４が、オブジェクトの軸心に位置づけられる1組のベクトルを単純に使用したとすれば、局部的な光源の動きは大きな局部的な照度変化を引起こさないが、全体としてオブジェクトのライティングに大きな影響を与えることもある。これらの状況においては、境界球の面におけるライティング変化のサンプリングを行えばオブジェクトに対するライティング変化をより正確に捕えることも可能であるが、それはオブジェクトの面上の特徴点において選択的に見た場合には達成され得ないものである。

【０２９５】

更なる代替例として、オブジェクトの特徴点、又は境界球の面における法線を組合せて使用し、ライティング変化をサンプリングすることも可能である。この試みによればライティング変化をより効率的に追跡することが可能である。なぜなら、オブジェクト上の特徴点で、及び、オブジェクトに対する境界ボリュームにおける面で、ライティング変化を追跡するからである。

【０２９６】

システムはライティング変化に基づき、これらのライティング変化を近似すべくｇスプライトのカラー値を改変する方法を決定する。ｇスプライトに関して行われる幾何形状変換と同様に、システムはｇスプライトのカラー値をワープしてライティング変化を近似する方法を計算する。カラーワープを計算するひとつの方法は、上述の最小２乗適合法を使用することである。このステップの結果は、定数、線形又は高次のワープとなり、これは、ｇスプライトに跨がる画素におけるカラー値を改変（例えば縮尺率を乗じ、及び／又は、オフセット値を加算）すべく用いられる。

【０２９７】

カラーワープは、ｇスプライトに跨がって適用される乗数又は多数の乗数を含んでいる。最も単純な場合、カラーワープは単純に、ｇスプライト内の全ての画素に適用される一定の縮尺率とできる。より正確なアプローチは、線形又は高次のワープを使用してライティング変化を近似することである。好適には、乗数はベクトルによる値とし、カラー成分を独立して換算することができるようにする。カラー光源からの変化を正確に表現する為

に、各カラー成分は独立して換算しなければならない。

【０２９８】

乗数に加え、ｇスプライト内のカラー値に加算されたオフセット値もまた、特徴点におけるライティング変化に基づいて計算することができる。

乗数及びオフセット値を計算するひとつの手法は、各特徴点におけるライティング式の変化を表す乗数及びオフセットに関し、特徴点がオブジェクトの面、境界ボリュームの面、又は、両者に位置しているかを解くことである。画像前処理装置１０４は乗数、オフセット、又は両者を計算することができるが、これは、サンプリングステージの間に観測された各特徴点のライティング式の同一な、又は実質的に同一な変化を引起こす乗数又はオフセット、又は乗数とオフセットの組合せを選択することにより計算される。これらの乗数及び／又はオフセットが一旦計算されれば、ｇスプライト内のカラー値に適用される乗数及びオフセットを計算する方法が多数存在する。ひとつの方法は、乗数を平均することにより、ｇスプライトに対する単一の縮尺率を導出することである。別の方法は、オフセットを平均することにより、ｇスプライトに対する単一のオフセットを導出することである。更に別の方法は、乗数及びオフセットに関して独立的に最小２乗適合法を行い、乗数及びオフセットがオブジェクトの面上の位置により如何に変化するかを表す式を導出することである。この式は、ｇスプライト内の画素位置に対して独立した乗数及び／又はオフセットを補間計算するハードウェアにて実現することができる。例えばｇスプライト・エンジンは、各画素位置に対する乗数及び／又はオフセットを補間する補間器を備えたラスタライザを含むことができるが、この補間は、カラー値に乗数を乗じ、又は、カラー値又は換算カラー値（即ち、画素位置に対して計算された対応乗数により換算された値）にオフセットを加算する前に行われる。

【０２９９】

システムは幾何ワープの精度を評価するのと同時に、カラーワープの精度も評価するが、これは、カラーワープにより計算されたカラー値を、通常のレンダリング処理を用いて現在のフレームに対して計算された対応カラー値と比較することで行われる。これらのカラー値が所定の許容差と異なるのであれば、ｇスプライトを再レンダリングせねばならない。

【０３００】

レンダリングの総量を減少することに加え、ｇスプライトのワープは伝達遅延(transport delay) をも減少させることができる。この点、遠近像の視点(viewpoint perspective)が急激に変化するアプリケーションにおいては、急激に変化しつつある視認図は伝達遅延の故に表示することが困難である。尚、伝達遅延とは、視点の変化を引起こす受信入力と、その新たな視点に対する適切な画像の最終的な表示との間に生ずる遅延のことを指している。図２３は如何にして伝達遅延が減少させるかの例を示している。水平軸心に沿った箇所は、フレーム遅延に対応する時的な経過を表している。

【０３０１】

この例においては、サンプリング入力と表示装置への表示出力との間には３フレームの遅延がある。最初に、入力は第１のフレーム774 でサンプリングされる。次に、システムはアフィン変換式を計算するとともにｇスプライト内にオブジェクトをレンダリングする(７７６)。最後に、フレームに対してレンダリングされた画像データは合成されて表示装置に走査出力される（７７８）。これらのステップの各々を実行するに必要とされる時間は、必ずしも、フレーム周期により測定された全体的なフレーム遅延ではないが、概念を示す為にフレーム周期の増加を用いている。図示される様に、対応する画像データにおける入力と表示との間には、３フレームの遅延がある。

【０３０２】

伝達遅延を減少する為に、次の画像からの視点データは現在の画像のレンダリングのフェーズで適用することができる（７７６）。これは、次の画像に対する入力・フェーズ７８２ から、現在の画像のｇスプライト変換及びレンダリングのフェーズ（７７６） への矢印により示されている。図２３に示される如く、画像データの次のフレームに対して処

理するフェーズ(７８２、７８４ 、７８０)は、フェーズ(７７６、７７８)の近傍に示されている。図示された様に、処理はパイプライン形式で生ずる。現在のフレーム対するｇスプライト変換が計算されるとともにレンダリングが行われる一方、次フレームに対する入力がサンプリングされる。

現在の画像のモデル変換を、ｇスプライト変換を計算するために、次の画像のビューイング変換と関連して用いることができ、通常、アフィン変換行列の形式で用いることができる。

而して、レンダリングされたｇスプライトは、次の画像の視点に対する位置をシミュレートすべくワープされる。この試みによればユーザに関する伝達遅延の影響を低減させる。なぜなら、それによりシステムは遠近像の視点における急激な変化をより迅速に調整することができるようになるからである。

【０３０３】

この関連で伝達遅延を減少するのに加え、次の画像データを使用することにより他の関連でも伝達遅延を減少することができる。

上記で概略的に述べた如く、ｇスプライトを独立的にレンダリングすることには多くの利点がある。各ｇスプライトは可変の更新速度を有していることから、特定のフレーム内で更新されるｇスプライトの個数も変化する。幾つかのｇスプライトはフレーム毎に更新が必要である一方、他のｇスプライトの更新頻度は更に少ないこともある。特定のフレーム内で多数のｇスプライトが更新されるべきであれば、レンダリング総量は劇的に増大してシステムが過負荷になり得る。この問題に対処すべく、システムは優先キューを作り、多数のフレームへレンダリングを配分してｇスプライトを一層効率的に処理することを可能としている。

【０３０４】

この優先キューを作らなければ、特定フレーム内でレンダリングが予定されるｇスプライトの数は変化してしまう。ｇスプライトに対する更新速度は、それが場面の前景であるか背景であるかに依って変化し得る。上述のアフィン・ワープに対するサポートにより、アフィン変換による位置における変化をシミュレートすることでシステムはｇスプライトの再レンダリングを回避することができる。

【０３０５】

優先キューを履行する為にシステムは、レンダリングされたｇスプライトの再使用により生ずる歪み量に基づいてレンダリングの優先付けを行なう。この歪みは、ひとつ以上のエラー閾値に基づいて計算される。また、ｇスプライトの歪みを定量化する為にシステムは、ｇスプライトがそのエラー閾値に如何に近いか又は如何に遠いかを測定する。エラー閾値は各ｇスプライト毎に変化するとともに、ひとつ以上の要因に基づいている。レンダリングを行なう前にｇスプライトの相対的品質を示すべく、ｇスプライトの歪み順リストが保持される。その場合、システムリソースを考慮して、可能な限り多数のｇスプライトがフレーム内で再レンダリングされる。ｇスプライトの再レンダリングの順番は、最も歪みの多いｇスプライトから始め、歪みの少ないｇスプライトへと行われる。この様にして処理を行えばｇスプライトのレンダリングによるフレーム過負荷の可能性が排除されるが、これは、ｇスプライトの精度に対して場面の複雑さ及び動きをバランスさせる為の効率的なメカニズムを提供する代わりに行われる。

【０３０６】

上述した特徴に加え、ｇスプライトのレンダリングによりシステムは場面内のオブジェクトの解像度を変更することができる。これにより、システムは場面におけるｇスプライトの重要性に基づいてそれらに処理及びメモリリソースを割り当てる。

ｇスプライトに対するコストは、それが占有するメモリと、それをレンダリングする為に必要な処理とに関して測定することが可能である。ｇスプライト画像が一定の解像度、即ち画面解像度で記憶かつレンダリングされるのであれば、ｇスプライトによって生じるコストはその画面の大きさにより決定される。

【０３０７】

処理及びメモリリソースの割り当ては、画面上におけるオブジェクトの単なる占有サイズではなく、その種類及び位置に基づくことが重要である。場面の前景において有効オブジェクトは、背景におけるものよりも場面に対して重要であるのが通常である。しかしながら、ｇスプライトがサイズに基づいて場面を割り当てられたなら、背景の画面は一層大きいことから処理及びメモリコストは一層大きくなる。

システムは、画面の解像度をｇスプライトの解像度から切り離し、ｇスプライトのコストをその最終的な画面範囲から独立させて設定することも可能である。また、システムはこれを達成する上で、ｇスプライトの解像度を適切に選択し、次に、ｇスプライトを適切なサイズに換算する。

【０３０８】

倍率又は縮尺率は、画像の画面上での大きさ及びｇスプライトの解像度から導出することができる。典型的には、グラフィックス・アプリケーションが画面の大きさを与える。また、グラフィックス・アプリケーションが解像度を指定する。代わりに、利用できるリソースと場面内のｇスプライトの相対的な重要度とに基づいて画像前処理装置がｇスプライト解像度を決定することも可能である。

【０３０９】

動作時に画像処理装置１０６は出力装置座標内の領域にｇスプライトをレンダリングするが、この領域は、ビュー空間内でｇスプライトが実際に占有するものよりも小さい。ｇスプライトがレンダリングされる領域のサイズは、解像度及び画面の大きさから導かれる。レンダリングされたｇスプライトは次に、画面の大きさにより規定される実際のサイズに換算される。ｇスプライトは小さな領域を有することから、レンダリングの為に消費するメモリ及びリソースは一層少ない。更に、図示された実施例においては、解像度が変化するｇスプライトもまた共通のグラフィックス・パイプラインで処理される。

【０３１０】

この試みをサポートするひとつの方法は、ｇスプライト・データ構造内に倍率又は縮尺率を記憶することである。次にこの縮尺率は、ｇスプライトが他のｇスプライトと合成されて表示画像を生成する前の換算に使用することができる。画像前処理装置はｇスプライトの換算を行なうことができる。より詳細には、上述の内容を実行する上では、ＤＳＰがｇスプライトを換算する。

ｇスプライトが低い解像度に換算することができる様に、それはひとつのサイズにレンダリングされてから小さな表示領域に換算することができる。この技術は、場面内でサイズが小さくなっていくオブジェクトに適用することができる。フレーム毎にオブジェクトをレンダリングする代わり、システムはオブジェクトを表すｇスプライトを換算することができる。この試みは、ｇスプライト・データ構造内に縮尺率も記憶することで行われる。

【０３１１】

上記では、画像処理システムによるｇスプライト処理を説明するとともに、ｇスプライト変換が如何にして計算されて画像処理システム内で適用されるのかを説明した。以下には、画素データを変換、合成及び表示する方法を詳述する。

この実施例においては、ＤＳＰ１７６はｇスプライト・データ構造を設定するとともに、それらを画像処理ボード１７４上の共用メモリ内に記憶する。ＤＳＰ１７６は、タイラーを通してメモリマップインターフェイスによりｇスプライト・エンジン・レジスタに読み書きを行なう。ｇスプライト・エンジン内のレジスタは、現在の表示リストへのポインタを含んでいる。また、ｇスプライト・エンジン４３６に関する詳細は、図１２を参照して既に説明されている。

【０３１２】

ｇスプライト・エンジン２０４ への主要な入力データは、ｇスプライト表示リストである。図２４は、表示リスト８００の一例を示している。これを実現する上で、表示リスト８００は、その各々がバンドマスク８０２を伴うＳＣＢ(ｇスプライト制御ブロック)ハンドル８０４と称されるｇスプライト制御ブロックアドレスのアレイを備えている。リ

スト８００内の最初のワードは、リスト内のｇスプライトの個数を含んでいる。バンドマスク内でビットがセットされていれば、このバンド内にｇスプライトが存在していることが示される。此処では特定の実施例が与えられているが、表示リストは別の手法に依っても実現することができる。例えば、リストは各バンドに対する別個のリストから成り、各バンドリストが当該バンドに影響するｇスプライトを列挙することも可能である。上述の如く、表示リスト内のｇスプライトは深度の順で記憶され、この場合は前から後への順で記憶される。

【０３１３】

ｇスプライト制御ブロック(ＳＣＢ)８０６は、ｇスプライトを走査して装置座標を出力する情報を含んでいる。矩形のｇスプライトはアフィン変換のもとでは画面空間内に平行四辺形としてマッピングされる。

ｇスプライトのエッジ方程式は次の形態を有している：$A_0x + B_0y + C_0=F_0$; $A_1x + B_1y + C_1=F_1$; $-A_0x + B_0y + C_2=F_2$; $-A_1x - B_1y + C_3=F_3$。これらの式の各右辺は、それぞれのエッジでゼロである。ＤＳＰ１７６は、ｇスプライトに対するアフィン変換から係数の値を決定する。アフィン変換の後、ｇスプライトの形状は平行四辺形であることから、係数Ａ 及びＢ の２組のみの記憶が必要とされる。Ｃ 項は全く必要とされない、と言うのも、ｇスプライト・エンジンが必要とするのは始点におけるＦの値と、画面空間Ｘ 及びＹ内でのステップによりＦの値が如何に変化するのかの記述であり、それは係数Ａ 及びＢ により与えられるからである。出力装置座標への記憶ｇスプライト・データのマッピングをサポートする為に係数の符号が設定されるが、それは、平行四辺形内の点の座標がエッジ方程式で評価されたときに結果が正の数になる如く行われる。

【０３１４】

詳細には、ＳＣＢ が含むのは：A_0、B_0;A_1 、B_1; F_0、F_1、F_2、F_3; 最左点 xs, ys;最右点 xf, yf; Ｇスプライトの頂部への最左点の勾配；底部への最左点の勾配；及び、平行四辺形の幅及び高さである。

走査の始点は平行四辺形の最左点であり、走査は画面空間内で左から右へ向かって列毎に移動する。ｇスプライトを各32走査線の画面バンドにクリップする為に、ＳＣＢは始点（最左点）から頂部及び底部への勾配dx/dy を含むことから、特定の画面バンド上の最左点を決定することができる。

【０３１５】

平行四辺形のエッジ方程式はＤＳＰ１７６上で正規化されるが、これは、平行四辺形のひとつの辺にて「Ｆ＝０」となり、対向辺にては「Ｆ＝ｇスプライトの幅又は高さ」となる如く行われる。従って、平行四辺形の辺０ 及び１に対するＦの値は、特定の画面位置Ｘ、Ｙにおける特定のｇスプライトの画像サンプルＳ、Ｔを直接的に調べるために使用することが可能である。画面Ｘ、Ｙ からｇスプライトＳ、Ｔへのマッピングがｇスプライトの画像サンプル上に直接的に落とされることは稀であることから、ｇスプライト・エンジンは直近の４個の（又は１６個の）ｇスプライトの画像サンプルを補間して出力サンプルを求める。

【０３１６】

ＳＣＢ ８０６は、元のｇスプライトのサイズ（水平及び垂直ストライド）と、走査すべきサブｇスプライトのサイズ及び位置（幅、高さ、開始Ｓ及びＴ）とを含む。それはまた、画像チャンクが如何に圧縮されるとともにこの画像塊内でどの画素フォーマットが使用されたのかを記述するフラグも含めることができる。

【０３１７】

チャンキング・アーキテクチャにおいては、ｇスプライトは、３２×３２画素チャンクに分けられる。再生のためにｇスプライトをチャンクにする必要はない。しかしながら、チャンキング・アーキテクチャは、上述したように、多くの利点を有している。チャンキング・アーキテクチャを支援するために、ＳＣＢは、圧縮チャンクの第１語用の共用メモリにおけるアドレスを表示する圧縮チャンクの二次元アレイのポインタ（チャンクハンドル）を有している。チャンクメモリは、５１２ビットのブロックで処理される。各ポイン

タ、即ちチャンクハンドルは、１８ビットであり、全部で16MBのアドレス指定可能なメモリとできる。各チャンクを圧縮するのに必要なメモリ量は可変であるため、各５１２ビットのブロックは、次のブロックのための１８ビットポインタを含んでいる。もはや不要となったブロックは、解放されたブロックのリンクされたリストに追記されるので、これらブロックは、別のチャンクで使用可能である。

【０３１８】

ｇスプライトに割り当てられたオブジェクトをチャンクに分割すると、ｇスプライト・データ構造は更新され、ｇスプライトの画像データを含むチャンクに対する参照が含まれるようになる。

ｇスプライト・データは、別のｇスプライトで例示できる。図２０で示した例では、１つのｇスプライトが別のｇスプライトからの画像データを例示している。ここで、ＳＣＢの第１チャンクハンドル（８０８）が別のｇスプライトのＳＣＢ８１０を指している。別つの方法では、チャンクハンドルは、チャンクが記憶されているメモリ中の位置を云う。

第２５図は、２個のチャンクｇスプライトによる６個のチャンクが表示の水平バンドにマップする態様の例である。図２５は、画像データをｇスプライト空間から物理的出力装置空間まで走査に使用される始点８３６と終点834を示している。ｇスプライト画像データがどのようにして出力装置空間にマップされるかを以下に詳細に説明する。

【０３１９】

フレーム中のｇスプライトの関連変化をレンダリング及び計算した後、画像処理装置１０６は、表示を行なう。図２１で示すように、画像処理装置１０６は、ｇスプライトを物理的出力座標に変換してｇスプライトを合成する。画素データを合成後、画像処理装置１０６は、これを表示に転送する。

【０３２０】

本実施例では、ｇスプライト・エンジンは、表示リスト中で読み、ｇスプライト画像を出力装置座標にマップする。ｇスプライト・エンジンがｇスプライト・データを変換し、ついで表示のために画素データを合成バッファに送る。合成バッファは、好ましくは、ダブルバッファとして、画素データを一方のバッファで合成している間に、合成画素データを他方のバッファから変換できるようにする。

さらに具体的には、ｇスプライト・エンジンは、共用メモリからのｇスプライトＡＹＵＶフォーマットの画像データを読み、復元、変換し、且つフィルタリングしてＡＲＧＢフォーマットに変換する。さらにこのデータは、ビデオレート（例えば、７５Ｈｚ）で合成バッファに送られる。合成バッファは、復元されたＡＲＧＢ画素を１３３４×３２フレームバッファに合成する。

【０３２１】

図２６は、ｇスプライト・エンジンが画像データを処理する態様を示している。フレーム同期信号を受信時（８５８）、ｇスプライト・エンジンは、各バンドを通してフレームをループ処理し、バンド中で各ｇスプライトを走査する（８６２）。バンドにおいてｇスプライトを走査後に次のバンドに移動する（８６０）。ｇスプライト・エンジンは、ビュー空間の各バンドでの走査を繰り返す。

【０３２２】

リアルタイム・アプリケーションでは、ｇスプライト・エンジンは、フレームレートで定まる時間内に走査を完了しなければならないので、ｇスプライト・エンジンは、バンドごとに全てのｇスプライトを処理できなくても良い。このような場合を回避する手だてとして、ｇスプライト・エンジンは、各バンド用のフリーな処理時間に各フレームをホストに戻す。この情報を利用して画像前処理装置は、必要に応じてオブジェクトを統合させて、いずれの個々のバンドについてもオーバーロードになるのを防いでいる。

【０３２３】

ｇスプライト空間からの画素を走査するにおいて、ｇスプライト・エンジンは、画素データを出力装置座標（８６６）へ変換する。従来の多数の走査技法のいずれをもｇスプライトを走査して出力装置座標に走査出力するに使用可能である。逆方向及び順方向のいず

れのマッピングも可能である。ｇスプライト・エンジンは、本実施例では、逆方向マッピング法を利用している。

ＳＣＢのエッジ方程式のデータを使用して、ｇスプライト・エンジンは、ｇスプライトをクリップして各バンドにおける走査の開始する位置を決定する。例えば、図２５は、ｇスプライトのエッジが３番目のバンド（８３０，８３２）に如何に跨るかを示している。交差点がこの個々のバンドにおけるｇスプライトの始点と終点となる。１つの走査法では、始点からジグザグなパターンで走査する。バンドの始点は、出力装置座標における交差点に最も近接する画素を導き出すことにより見つける。始点が演算されれば、ｇスプライト・エンジンは、ｇスプライトの外側、又はバンドの外になるまで始点を増加方向に上方にステップする。次いで１画素列分、右へステップし、ｇスプライトの外側、又はバンドの外になるまで下方にステップする。各ステップで、ｇスプライトの画素データを補間し、各ステップでの画素位置の画素値を見い出す。各位置での画素値を演算した時に合成バッファに画素データを送り合成をする。

【０３２４】

図２７は、ｇスプライト・エンジンと合成バッファが画像データのバンドを処理する態様を示している。この図では、“バンド”なる語は、画素データのバンドを処理するために割り付けられた時間の量（バンド周期）に関するものである。図２７に示すように、ｇスプライト・エンジン２０４は、あるバンド用の合成バッファ２１０を満たし（８８８）、これにより合成画像データが表示装置に走査出力される（８９２）。ダブルバッファを利用することで、これらステップは、続くバンドでオーバーラップされる。ｇスプライト・エンジン２０４は、あるバンドによって合成バッファを満たしている間に（８９０）、その合成バッファは、別のバンドの合成画像データをＤＡＣ２１２に転送する（８９２）。次のバンド周期で現在合成されたばかりのバントが表示装置に表示される（８９４）。表示上の各バンドに対して、この処理を繰り返す。このダブルバッファにより画素を変換し、合成する処理は、バンドを表示する処理と同時に行なうことができる。

【０３２５】

ｇスプライトとは、リアルタイムで合成でき、出力装置で表示される画像を生成させる。合成バッファは、２つの３２走査線バッファを有し、その一方は、合成に使用され、他方は、表示のためのビデオデータを生成させるために使用される。この二つのバッファは、交互にやり取りされて１つの走査領域が表示されている間に次の走査領域が合成される。

ｇスプライト・データは、合成すべき各画素につき一次カラー値とアルファ値とを合成バッファに転送する。32走査線アルファ・バッファは、合成に使用される走査線バッファと関連させる。ｇスプライト・データは、前後の順に処理されるので、アルファ・バッファは、各画素の不透明度を蓄えて、正しいアンチエイリアシング及び半透明処理を可能とする。

【０３２６】

走査線カラーバッファは、０．０（全ビットをリセット）に初期化され、アルファ・バッファは１．０（全ビットをセット）に初期化される。各画素につき、走査線バッファにロードされたカラー値が計算される。即ちcolor(new) = color(dst) + color(src) * alpha(src) * alpha(dst)。アルファ・バッファに記憶されたアルファ値は、alpha(new) = alpha(dst) * (1 minus alpha(src))により計算される。好ましくは、カラールックアップテーブル（ＬＵＴ）は、２５６ｘ１０ビットである。残りのビット（１０ｖｓ８）は、より正確なγ補正を設けるように用いることができる。

【０３２７】

タイリング

上述したように、画像処理装置（図１）は、走査－変換を行い、さらに隠れ面を消去し、アンチエイリアシング、半透明性計算、テクスチャリング、シャドウイングが行われる。本章では、走査変換、隠れ面の消去及びアンチエイリアシングの透明度の演算を詳細に説明する。

【０３２８】
図４Ｂは、幾何形状プリミティブからのレンダリングされた画像を生成する画像処理装置４６２の部分を示すブロック図である。本実施例では、画像処理装置４６２は、ラスタライザ４６４、画素エンジン４６６、アンチエイリアシング・エンジン４６８、画素バッファ４７０を含むラスタライゼーションバッファ及びフラグメントバッファ４７２を含む。“ラスタライザ”は、例えば、ポリゴンなどの幾何形状プリミティブから画素値を決定する画像処理装置の部分に関するものである。ラスタライザ４６４は、プリミティブデータを読み取り、画素位置に関連した画素値を出力する。この画素データには、カラー値、アルファ値及び深度値（視点からの距離）を含む。画素がポリゴンのみに完全にカバーされない場合は、ラスタライザは、画素フラグメントデータを生成する。

【０３２９】
ポリゴンの変換時、ラスタライザは、画素データを画素エンジンに転送して処理する。画素エンジン４６８は、ラスタライザからの画素データを読み取り、どの画素データを画素バッファやセグメントバッファに記憶させるかを判断する。画素バッファ４７２は、二次元アレイであり、アレイ要素は、画素位置に相当し、カラー、アルファ及び深度データを記憶するメモリを含んでいる。フラグメントバッファ４７０は、フラグメントデータを記憶し、画素の部分的カバーを与える。

【０３３０】
画素エンジン４６６は、ラスタライザで生成された深度値を利用して隠れ面の消去を行い、またアンチエイリアシングと半透明処理のための画素フラグメント及び不透明性の画素を保持する。所与の画素位置に対して、最も近接する、完全にカバーされた不透明性の画素があれば、これを保持する。この点に関して、“完全カバー”は、走査され、ラスタライザにより変換されたポリゴンによって、完全にカバーされる画素を意味している。また画素エンジンは、透明度（１＞アルファ）を有する画素と画素フラグメントを、最も近接する不透明性の画素の前方に保持する。画素エンジンは、画素バッファに、画素位置に関しての最も近接する不透明性の画素を記憶し、フラグメントバッファには、最も近接する不透明性の画素の前方にある、この画素位置で、いずれかの画素フラグメント又は透明性の画素を、フラグメントバッファに記憶する。

【０３３１】
画素エンジンが画素データを生成後、アンチエイリアシング・エンジン４６８は、画素及びフラグメントバッファ内の画素データを解析する。図４Ｂに示した画像処理装置の構成は、画素データのダブルバッファとフラグメントデータの単一バッファの構成となっている。画素エンジンは、画素データを画素バッファの１つで生成し、フラグメントバッファにフラグメント情報を追加し、この間にアンチエイリアシング・エンジンは、別の画素バッファからの画素データ及びフラグメントバッファからのフラグメントバッファを解析する。各フラグメントが解析されると、フラグメント・エントリが、新たな画素データにより使用されるフラグメント・フリーリストにエントリされる。

【０３３２】
画素データを生成及び解析するプロセスを概略したが、実施例を詳細に説明する。
図４Ｂのコンポーネントは、タイラーを充足するものである。タイラーは、共用メモリシステム２１６（図４Ａ）から、プリミティブデータならびにレンダリング指示を読み取り、レンダリングされた画像データを生成し、共用メモリに圧縮画像データを記憶する。上述したように、システム中の基本的な３Ｄグラフィックプリミティブは、三角形である。三角形は、常に平面でかつ凸であるので、三角形のレンダリングは、グラフィックス生成に使用されるハードウェアにおいて数々の簡素化を与えるものである。しかしながら、この代わりに、ｎ片のポリゴンを使用することもできる。

【０３３３】
タイラー２００のコンポーネントについては上述した。次にタイラー内のデータの流れを説明する。
タイラーは、ＤＳＰからの入力値を受け取るので、ＤＳＰ１７６（図４）の機能の説明

から始める。上述したように、ＤＳＰ１７６は、フロントエンド幾何形状を実施し、３Ｄグラフィックに要されるライティング計算を行なう。ＤＳＰ１７６は、モデル変換及びビューイング変換、クリッピング、ライティングなどを行なう。レンダリング命令は、メインメモリ・バッファに記憶され、ＰＣＬバス上で画像処理ボードにＤＭＡ（ダイレクトメモリアクセス）される。レンダリング命令は、次いで共用メモリ２１６（図４Ａ）にＤＳＰで必要とされまで一時的に記憶される。レンダリング命令は、画像処理動作を行なう準備が整った時に、タイラー２００（図４Ａ）により読み取られる。

【０３３４】

図２８Ａと図２８Ｂのフローチャートに示すように、セットアップブロックは、共用メモリから読み取られたプリミティブのレンダリング指示を処理する。頂点入力処理装置は、入力ストリーム（９１４）（図２８Ａ）を解析し、頂点制御レジスタ（９１６）にプリミティブの三角形処理に必要な情報を記憶する。

二つの頂点制御レジスタが、それぞれ３個ずつ、６個の頂点を記憶する。二つの頂点制御レジスタは、三角形情報のダブルバッファリングを可能としてセットアップ・エンジンが常に処理する三角形情報を有するようにする。

セットアップ・エンジンは、次いでエッジ、カラー及び三角形の面を跨るテクスチャ座標の補間を決定する線形方程式を計算する（９１８）。これらの線形方程式は、どのテクスチャのブロックがプリミティブの三角形処理に必要かを判断するものである。エッジ方程式は、走査変換ブロックに通され（９２０）、走査変換エンジンが必要とするまで走査変換ブロック内のプリミティブレジスタに記憶される。プリミティブレジスタは、多数の組のエッジ方程式を記憶可能である。

【０３３５】

セットアップ・エンジンは、テクスチャアドレスを、テクスチャチャンク用の要求をバッファするテクスチャ読み取りキューに転送する（９２２）。テクスチャアドレスレジスタは、次いで、要求されたテクスチャチャンクのメモリ中のアドレスを決定し（９２４）、テクスチャ読み取り要求を、命令・メモリ制御装置に送り（９２６）（図２８Ｂ）、走査変換ブロックにより使用されるテクスチャデータをフェッチする（９２８）。

【０３３６】

テクスチャデータは、画像フォーマットと同一のフォーマットでも良い圧縮画像フォーマットで共用メモリ（２１６）に記憶される（図４Ａ）。圧縮フォーマットは、個々の８×８画素ブロックでなされる。８×８のブロックは、メモリ管理オーバーヘッドを低減するためのメモリ管理の目的で、３２×３２のブロックにグループ化される。

【０３３７】

テクスチャブロックが必要とされる時に、これらは、タイラーにフェッチされ、復元エンジンにより復元され（９３０）、オンチップ・テクスチャキャッシュでキャッシュされる（９３２）。各ブロックがただ１つのカラー成分を記憶可能であったとしても、総じて３２８×８画素ブロックがキャッシュ可能である。テクスチャデータは、ＲＧＢ及びアルファのフォーマットでキャッシュされる。

走査変換エンジンは、プリミティブレジスタからエッジ方程式を読み取り（９３４）、三角形エッジ情報を変換する。走査変換エンジンは、三角形のエッジを進み、カラー、深度、透明度などを補間する補間器を含む。

走査変換エンジンは、テクスチャアドレスをテクスチャフィルタ・エンジンに転送する（９３６）。テクスチャフィルタ・エンジンは、レンダリングされているポリゴン用のテクスチャデータを計算する。テクスチャフィルタ・エンジンは、Ｚ－スロープ、三角形の原点、ｓ及びｔ座標の基づくフィルタ・カーネルを演算する。テクスチャフィルタ・エンジンに付随するテクスチャキャッシュは、１６個の８×８画素ブロック用のデータを記憶する。テクスチャキャッシュは、復元エンジンにも連通している。この復元エンジンは、テクスチャフィルタ・エンジンで使用するために、テクスチャデータ（圧縮状態で記憶されている）を復元する。

【０３３８】

テクスチャフィルタリングが完了すると、テクスチャフィルタ・エンジンは、情報を走査変換エンジンに戻し（938）、さらなる処理のために走査変換エンジンにより使用可能である。テクスチャ処理の過程で、走査変換エンジンは、三角形エッジデータを変換し（940）、カラー及び深度情報を含む個々の画素アドレスが画素エンジンに転送され処理される（942）。

図２８Ａ及び図２８Ｂに図示の方法は、図１０と図１１に関連して説明した代替方法に関して変化する。図２８Ｃ及び図２８Ｄは、図１０と図９Ｂに相当する画像データをアクセスする方法を示している。同様に、図２８Ｅと図２８Ｆは、図１１と図９Ｃに相当する画像データをアクセスする方法を示している。

【０３３９】

図２８Ｃと図２８Ｄを参照して、この方法の実行は、図９Ｂのセットアップブロック３８１で始まる。頂点入力処理装置３８４は、入力データ・ストリームを処理する（９４７）。次に、頂点制御レジスタ386は、入力データ・ストリームからの三角形データをバッファする（９４８）。セットアップ・エンジン３８８は、次いでエッジ方程式を計算し（９４９）、この結果を走査変換ブロック395に転送する（９５０）。

【０３４０】

走査変換ブロック３９５は、プリミティブレジスタに記憶されたエッジ方程式を読み取り（９５１）、三角形データを走査変換する（９５２）。走査変換エンジン３９８は、次に画素アドレス、カラー及びアルファのデータならびにカバレッジデータを含む画素データをテクスチャ参照データ・キュー３９９におけるエントリに書き込む（９５３）（図２８Ｄ）。テクスチャ・マッピング動作の場合において、このエントリは、テクスチャ参照データ、即ちテクスチャの中心点の座標を含む。エントリは、またレベルデータや異方性フィルタのデータを含んでも良い。

【０３４１】

テクスチャ参照データから、テクスチャキャッシュ制御装置３９１は、どのテクスチャブロックをフェッチするかを判断し、メモリから適正のテクスチャブロックをフェッチする（９５４）。テクスチャアドレスキャッシュ制御装置３９１は、テクスチャ読み取り要求を命令・メモリ制御装置３８０に転送する（９５５）。テクスチャ読み取りキュー３９３は、テクスチャブロック用の読み取り要求を共用メモリシステムにバッファする。命令・メモリ制御装置３８０は、共用メモリからのテクスチャデータをフェッチし、テクスチャデータが圧縮されていれば、その圧縮ブロックを圧縮キャッシュ４０２に配置する（９５７，９５８）。図１０に関して上述したように、テクスチャキャッシュ中のブロックの置き換えは、キャッシュ置換アルゴリズムに従って進行する。

【０３４２】

テクスチャキャッシュ中の画像データを要求するテクスチャ・マッピングあるいは別の画素動作を実行するために、テクスチャフィルタ・エンジン４０１は、テクスチャキャッシュ４０２に中の画像データにアクセスし、テクスチャのコントリビューションを計算し、このコントリビューションをテクスチャ参照データ・キュー３９９（９５５）からのカラーデータ及び多分アルファデータと組み合わせる。

テクスチャフィルタ・エンジン４０１は、画素データを画素エンジン４０６に転送し、これにより画素エンジンは隠れ面の消去を行い、ラスタライゼーションバッファへの画素データの記憶の制御をする。

【０３４３】

図２８Ｅ及び図２８Ｆは、図１１の方法に相当するメモリからの画像データのアクセスする方法を示している。この代替方法の実施においては、セットアップブロック３８３のプリミティブをキューすることから開始される。頂点入力処理装置３８４は、入力データ・ストリームを解析し、頂点制御レジスタ３８７の三角形データをキューに入れる（９６１，９６２）。テクスチャ・マッピング動作の場合と同様に、画像データブロックをメモリからアクセスする必要があれば、プリラスタライザ３８９は、頂点制御レジスタ３８６でキューされるプリミティブを走査変換し、共用メモリ９６３内のテクスチャデータブロ

ックに対する読み取り要求を生成する（９６３）。
プリラスタライザがセットアップブロックでキューされるプリミティブを走査すると、テクスチャ読み取り要求がテクスチャキャッシュ制御装置３９１に転送される（９６４）。テクスチャキャッシュ制御装置３９１は、適正テクスチャブロックを決定し（９６５）、読み取り要求をテクスチャ読み取りキュー３９３を介して命令・メモリ制御装置380に転送する（９８９）（図２８Ｆ）。命令・メモリ制御装置３８０は、要求されたテクスチャデータをフェッチし、テクスチャデータが圧縮されていれば、圧縮キャッシュ４１６にこのテクスチャデータを記憶する（９９０）。復元エンジンは、圧縮キャッシュ４１６内のテクスチャブロックを復元して、その復元したデータをテクスチャキャッシュ４０２に書き込む（９９１，９９２）。テクスチャキャッシュ制御装置は、復元エンジン４０４からテクスチャキャッシュ４０２に向けて、圧縮キャッシュ４１６からのテクスチャブロックの流れを管理する。

【０３４４】
走査変換ブロック３９７は、セットアップブロックでキューされる幾何形状プリミティブを読み取る。走査変換ブロック３９７は、テクスチャキャッシュ４０２で要求されたテクスチャデータが利用可能であれば即座に画素生成動作を実行する。これら画素動作の処理において、走査変換ブロック３９８は、プリミティブレジスタからエッジ方程式を読み取り（９９３）、テクスチャアドレスをテクスチャフィルタ・エンジン４０３に転送する（９９４）。テクスチャフィルタ・エンジンは、テクスチャキャッシュ４０２中に記憶された適正画像データにアクセスし、次いでフィルタされたデータを走査変換ブロック397に戻す（９９５）。走査変換ブロック３９７は、三角形データを変換し、変換された三角形データ及びフィルタリングされたデータから、出力画素データを演算する（９９６）。次いでこの出力画素データを画素エンジン４０６に送る。

【０３４５】
画素エンジン４０６は、隠れ面の消去及び混合動作を含む画素レベルの計算を実行する。隠れ面の消去を実行するため、画素エンジン４０６は、入力してくる画素（完全にカバーされた画素あるいは画素フラグメント）用の深度値と、画素又はフラグメントのバッファ内の、対応する位置での画素とを比較する。シャドウイング動作では、画素エンジン４０６は、シャドウマップ内の位置での光源に対して最も近接する第１と第２のプリミティブを決定するように深度比較動作を実行し、必要に応じてその最も近接する第１と第２のプリミティブを更新する。画素レベル計算の実行後に、画素エンジンは、画素あるいはフラグメントバッファに適正データを記憶する。

【０３４６】
タイラーは、非不透明性の画素を処理する高品質のアンチエイリアシング・アルゴリズムを実行する。画素バッファは、チャンク内の画素位置に対する最前部の非透過性の画素用の画素データを記憶する。フラグメントバッファは、半透明性の画素用の画素フラグメント、及び、対応する位置における画素バッファ内の画素よりも視点に近い、部分的にカバーされた画素用の画素フラグメントを記憶する。１つの画素位置に関して１つ以上のフラグメントがフラグメントリスト構造を利用して記憶することができる。解析と称される処理において、アンチエイリアシング・エンジンは、フラグメントリストを処理して画素位置に関するカラー値とアルファ値とを演算する。

【０３４７】
生成されるフラグメント数を減らすために、画素エンジンは、生成中のフラグメントと、現時点のバッファに記憶されているフラグメントとを比較し、それら画素フラグメントをマージする方法を実現する。新・旧のフラグメントのコントリビューション（カラー値と深度値）が、既定の許容範囲で同等であれば、それらフラグメントはフライ上で組み合わされ、別のフラグメントが生成されることはない。
組み合わされたフラグメントが、完全にカバー（完全カバレッジマスクと不透明度を表すアルファ値とともに）されたものと判断されると、そのフラグメントは、カラーバッファに書き込まれ、現時点のチャンク内の続くポリゴンに使用されるまで、そのフラグメン

ト位置はフリーにされる。

一旦、チャンクの全てのポリゴンがレンダリングされると、画素バッファはスワップされる。一方、アンチエイリアシング・エンジンは、フラグメントバッファの画素データと画素バッファの画素データを解析している際に、画素エンジンは、後続するチャンクの画素データを他の画素バッファに書き込み、また残りのフリーの位置をフラグメントバッファに書き込む。一般に、画素解析は、画素及び位置相当フラグメントバッファ中における画素データに基づく画素位置の単一のカラー値（及び、可能であれば、アルファ値も）を計算することを含んでいる。これらについて以下に詳細に説明する。

【０３４８】

図９Ａ～９Ｃに示すタイラーの実現例において、画素エンジン及びアンチエイリアシング・エンジンは、単一のフラグメントバッファと一組の画素バッファにアクセスする。２つの３２×３２画素バッファは、画素エンジンとアンチエイリアシング・エンジンとの間のダブルバッファとして、設けられる。画素バッファ・エントリは、以下のデータを包含する。

【０３４９】

【表２】

R	G	B	α	Z	P

【０３５０】

ここでＲ、Ｇ、Ｂは、赤、緑及び青のカラー成分であり、αは、画素の半透明性を表すアルファ成分であり、Ｚは、眼点からの画素の深度を表す深度成分である。ｘ、ｙアドレスは、固定アドレスであり、画素バッファにおける暗黙アドレスである。カラー成分（即ち、赤、緑、青）毎に、８ビットが用いられ、アルファ成分に８ビットが用いられ、Ｚ値、ステンシル値及び優先値に、２６ビットが用いられる。２６ビットのうち、２４ビットまでが、Ｚ値として用いられ、３ビットまでがステンシル面として用いられ、３ビットまでが優先値として用いられる。図９に関して上述したように、バッファは、また９ビットのフラグメントバッファポインタを含んでいる。

【０３５１】

優先値は、プリミティブごとに固定され、Ｚ比較動作中記憶されたＺ値と比較して入力してくる画素のＺ値をマージすべくタイリングエンジンにより用いられる優先関係を利用して、地形頂部の道筋など、共通平面上のオブジェクトの解析を助ける。

フラグメントバッファは、エッジが所与の画素と交差するポリゴンについての画素フラグメントの情報、あるいは半透明性を有するポリゴンについての画素フラグメントの情報を記憶するのに使用される。フラグメントバッファの各エントリは、カラー、α、Ｚ及びポリゴンの面と関連するカバレッジデータを提供する。

【０３５２】

複数のフラグメントバッファ・エントリは、ポリゴンが同じ画素位置で部分的カバレッジを有している場合のための単一画素と関連づけられる（連結されたリストのメカニズムによって）。フラグメントバッファは、デュアルポートであり、アンチエイリアシング・エンジンと画素エンジンの双方で並行に動作可能である。一実現例において、フラグメントバッファは、フラグメント記録の一次元アレイであり、総じて５１２個のフラグメント記録エントリを含む。フラグメントバッファのメモリ管理は、連結されたリスト構造を用いて実行される。各フラグメントバッファ・エントリは、以下のデータを含んでいる。

【０３５３】

【表３】

R	G	B	α	Z	M	P	S

【０３５４】
ここでＲ、Ｇ、Ｂは、赤、緑、青のカラー成分であり、アルファは、画素の半透明性を表すアルファ値であり、Ｚは、眼点からの画素の深度を示すＺ値であり、Ｍは、部分的にカバーされた画素用の４ｘ４画素カバレッジビットマスクであり、Ｐは、次のフラグメントバッファ・エントリに対するポインタであり、Ｓは、フラグメント・ステンシルを示すために用いられる。８ビットがカラー成分（即ち、赤、緑、青）ごとに用いられ、８ビットがアルファ成分ごとに利用され、２６ビットがＺ値＋ステンシルと優先度とを記憶するために用いられ、９ビットがフラグメントポインタのために用いられる。
【０３５５】
画素カバレッジマスクは、各エッジのカバレッジマスク値を決定し、これらをビット方向にＡＮＤをとることにより演算される。カバレッジマスクの演算は、２ステップの処理で行われる。第１ステップで、カバレッジマスク中のサブ画素のいくつがオンになるかを判断し、第２ステップでどの特定ビットが有効であるかを決定する。
【０３５６】
第１ステップは、カバレッジマスクのビットがいくつスイッチオンされるべきかを決定するために、エッジによってカバーされる画素の領域に使用される。この画素の領域は、エッジスロープと画素中心からの距離でインデックスされたルックアップテーブルにより演算される。第２ステップは、サンプルビットがスイッチオンされるべきビット順を決定するために、エッジスロープを利用する。１組のビット順が、“カバレッジ順序”テーブルと称する、予め計算されたテーブルに記憶させる。各カバレッジ順序テーブルのエントリは、スロープ値の範囲として正確なサンプルビットの特定順序からなる。エッジスロープは、スロープ範囲の組に対してテストされ、このスロープ値を含む範囲に関連するインデックスは、カバレッジ順序テーブルへのインデックスとして使用される。
【０３５７】
カバレッジマスクの演算方法は、Schilling, A. の“サブ画素マスク付きの新単純有効アンチエイリアシング”Computer Graphics 第２５巻、Ｎｏ．４、１９９１年７月、第１３３頁から第１４１頁、に記載されている。
【０３５８】
隠れ面の消去及びフラグメントのマージ
画素エンジンが、入力してくる画素データについて深度比較動作を行なうことにより隠れ面の消去を行なうことを上述した。また画素フラグメントがマージされ、フラグメントメモリをフリーにすることも上述した。フラグメントのマージは、所与の場面を、アンチエイリアシングを行なうための記憶要求を低減し、最終画像を生成するためのフラグメント解析の速度を上げる。ここで、入力してくるフラグメントが、記憶されたフラグメントの所定のカラー値と深度値の許容範囲内にある場合に、入力してくる画素フラグメントを、記憶された画素フラグメントとマージすることを含む、隠れ面の消去の実現例について説明する。
【０３５９】
図４Ｂは、ラスタライザ４６４と、画素エンジン４６６と、画素及びフラグメントのバッファ４７０及び４７１とを含むタイラー４６２内の構成要素を図示するブロック図である。画素及びフラグメントのバッファは、選定した画素データを記憶するラスタライゼーションバッファとして機能する。ラスタライザは、幾何形状プリミティブを跨って走査する際に、画素データ例を生成する。画素エンジンは、Ｚバッファの動作を制御し、さらに入力してくる画素フラグメントが、対応する画素位置でフラグメントバッファに記憶された画素フラグメントとマージすることが可能かどうかを決定する。図９Ａ～９Ｃに示すタ

イラーならびにその関連説明は、タイラーの特定の実施例に関する更なる詳細を与えるものである。以下に説明する画素フラグメントのマージする方法並びにそのハードウェアは、これらタイラー構成及び同様な構成において十分に実現できるものである。

上述のように、タイラー内の走査変換ブロック（ラスタライザ）は、１）完全にカバーされた不透明性の画素、２）完全にカバーされた半透明性の画素、３）部分的にカバーされた不透明性の画素、４）部分的にカバーされた半透明性の画素、を表す画素データ例を生成する。

画素バッファは、最前部の完全にカバーされた不透明性の画素のカラー及び深度（Ｚ）を記憶する。また画素バッファは、完全にカバーされないカバレッジマスクあるいは、完全に不透明とならないアルファ値を有するフラグメントを含む、フラグメントリストに対するポインタを記憶する。フラグメントリスト中の先頭（即ち、第１フラグメント）は、最も近時に処理された画素フラグメントである。この特定の実施方法では、画素エンジンは、入力してくる画素フラグメントと最も近時に処理された画素フラグメントとをマージすることを試みる。ポリゴンをレンダリングする際に、ある量の特殊なコヒーレンスがあるので、所与の画素位置に対して生成された最近のフラグメントとマージするのを試みることは、マージする成功度を増大させる。

【０３６０】

各画素位置に対するフラグメントリストは、ソートしてない形式で保持されている。また先頭フラグメントは、特定画素位置で生成した最近のフラグメントである。先頭フラグメントの背後の画素フラグメントは、ソートされていないが、フラグメント解析フェーズの最適化を助けるために、付加的な演算時間が可能であれば、ソートすることができる。

別の実施例において、画素エンジンは、フラグメントをマージする基準に合致する画素フラグメントをフラグメントリストで検索するために、付加的なロジックを含んでいる。この方法は、検索ロジックのオーバーヘッドがより高いマージ候補を認識する際に増加的な改良を生じさせることを正当化するものではないので好ましくない。これは特に、マージ処理で消費される追加のクロック・サイクルがアニメーションのフレームをレンダリングするに必要な時間を増大させる、リアルタイムシステムでは特にそうである。

【０３６１】

別の実施方法では、画素エンジンは、画素フラグメントの深度ソートリストを保持し、かつ所与の画素位置での視点に最も近接するフラグメントとのマージを試みる。この最後の方法は、しかしながら、好ましくない。なぜなら成功しうるマージ候補（即ち、入力してくるフラグメントに対して所定の許容範囲内のＺ値とカラー値を有するフラグメント）を見つけ出すことができそうもないからである。これは、付加的なフラグメントメモリのフリー化を簡素化するという潜在的利点を有している。マージされた画素が完全にカバーされ、不透明性であれば、マージされた画素が、記憶された他の画素フラグメントよりも視点により近い画素位置となるので、その画素位置での全ての画素フラグメントをフリーにすることができる。

【０３６２】

図２９は、タイラー内の隠れ面の消去とフラグメントをマージする実施例を示すフローチャートである。本処理は、まず画素位置でのカラー、Ｚ及びカバレッジマスクを有する新しい画素データ例の生成で始まる（９６８）。この画素位置での画素バッファにあるＺ値が、新しい画素データ例（完全にあるいは部分的にカバーされた画素）のＺ値より近ければ（９７０）、新しい画素データ例は、全く不明瞭なものであり、破棄される（９７２）。ラスタライザが、レンダリングされている現時点のプリミティブの組に対して、全ての画素が生成されない限りは、本処理は、さらに次の画素データ例で続けられる。

【０３６３】

新しい画素データ例のＺ値より画素バッファのＺ値が近いものでなければ（即ち、新しい画素データ例のＺ値が視点に対してより近い）、画素エンジンは、入力してくる画素に対するカバレッジマスクをチェックする（９７４）。入力してくる画素に対するカバレッジマスクが充分な場合は、画素エンジンは、画素バッファにあるカラー及びＺを、新しい

カラー及びＺに入れ換える（９７６）。この場合に、新たな画素データがフラグメントリストに追加されず、メモリが節約される。

【０３６４】

新しい画素データ例のカバレッジマスクが充分でなければ、画素エンジンは、マージテストを行い、新しいカラー値及びＺの値が、先頭フラグメントのカラー値及びＺの値についての所定の許容範囲内であるか否かを決定する（９７８）。このマージテストは、入力してくる画素のアルファ（半透明度）が、先頭フラグメント内の所定の許容範囲内にあるか否かを決定することを含めることができる。新しいフラグメントのカラー値及びＺの値が所定の許容範囲内にない場合は、新しい画素フラグメントを、フラグメントリストの先頭でフラグメントバッファに加えられる（９８０）。

【０３６５】

新しいフラグメントのカラー値及びＺの値が所定の許容範囲にあり、新しいカバレッジマスクが充分でない場合は、入力してきた画素フラグメントは、フラグメントリストの先頭フラグメントにマージされる（９８２）。画素エンジンは、先頭カバレッジマスク及び新しいカバレッジマスクについてＯＲ演算を行って画素フラグメントをマージし、結果を先頭カバレッジに残す。

【０３６６】

カバレッジマスクの組み合わせ後、マージされた先頭カバレッジマスクがチェックされ、それが完全にカバーされた画素を示すものか否かをチェックする（９８４）。マージされた先頭カバレッジマスクが充分でなければ、次の画素データ例で処理を継続する（９８６）。マージされた先頭カバレッジマスクが充分であれば、マージされた先頭カバレッジマスクは、完全な画素カバレッジを与える。よって、先頭フラグメントに使用される記憶領域は、フリーにされ（９８６）、先頭フラグメントカラー、Ｚ及びカバレッジマスクは、新しい先頭フラグメントのカラー、Ｚ及びカバレッジマスクに入れ換えられる（９７６）。

【０３６７】

画素エンジンが、画素バッファのエントリを新しい完全にカバーされた画素に入れ換える場合には、画素エンジンは、深度値がこの完全にカバーされた画素より大きい、対応するフラグメントリストにおける全ての画素フラグメントをフリーにする（９８８）。これは、入力してくる完全にカバーされた不透明性の画素が、同じ画素位置での画素バッファ・エントリよりも大きな深度値を有している場合に生じる。またこれは、マージされたフラグメントが完全にカバーされた不透明性のものであり、かつ同じ画素位置での画素バッファ・エントリより低いＺを有している場合に生じる。この場合には、画素エンジンは、フラグメントリストを縦断して、新しい完全にカバーされた画素のＺとリスト中のフラグメントのＺとを比較して、新しい完全にカバーされた画素のＺよりも大きいＺを有する各フラグメントをフリーにする。あるいは、Ｚバッファは、パッキング処理に対して省略可能であり、フラグメントリストの走査の必要性をなくし、リアルタイム性能を改良する。

【０３６８】

図２９に示す方法は、所与の場面をアンチエイリアシングするメモリ記憶要求を低減させ、使用されない画素フラグメントを破棄することにより最終のグラフィック画像を生成するためのフラグメント解析の速度を高めている。カラー値及びＺの値の許容範囲を調整することにより、生成したフラグメントについて破棄した数を、ユーザ要求に依存するアンチエイリアシングの精度に対して、バランスをとることができる。カラー値とＺの値を、画素センタに最も近いポリゴンのエッジで評価すれば、より狭いカラー値及びＺの値の許容範囲を使用でき、よりメモリ節約できる。

【０３６９】

図３０は、入力してくる画素フラグメントにおいて、マージテストを実行するように使用されるフラグメントマージ回路の一例を示すブロック図である。この例では、画素エンジンは、入力してくるカラー（Ｒ、Ｇ、Ｂ）、アルファ及び深度の各値を、入力してくる画素の位置に対して最近の画素フラグメントのカラー、アルファ及び深度の各値と比較す

る。“new”として表わされるカラー、アルファ及び深度の各成分は、入力してくる（即ち、“新しく”生成された）画素データ例を示し、“prev”と表わされた成分は、画素位置での最も近時に処理した画素フラグメントを示す。

画素エンジンは、フラグメントリストを縦断して、カラー及び深度の許容範囲内の画素フラグメントを見つけ出す場合の代替例では、マージテストを使用して解析される画素位置用のフラグメントリストにおいて、“prev”と表わされた成分が、各々の画素フラグメントを示すことになる。

【０３７０】

マージテストブロック１０００から１００８は、新しい画素フラグメントと前の画素フラグメントとの間で、深度、カラー及びアルファの各成分を比較し、新しい値と前の値が所定の許容範囲にある場合は、新しい画素フラグメントは、マージ候補であることを示すビットを出力する。画素エンジンは、次いでビット方向のＡＮＤ（１０１０）をとって各マージテストに合格したか否かを判定する。合格であれば、画素エンジンは、新しい画素フラグメントと前の画素フラグメントとをマージする。画素エンジンは、新しいカバレッジマスクと前のカバレッジマスクとのＯＲをとって前のフラグメントに対する新しいカバレッジマスクを演算する。いずれかのマージテストに合格しない場合は、画素エンジンは、新しい画素フラグメントをフラグメントリストの先頭に追加する。この新しいフラグメントは、リンクされたリストの一部となり、フラグメントリストの前の先頭を云うことになる。

【０３７１】

図３１は、 マージテスト・モジュールをさらに詳細に図示するブロック図である。 マージテスト・モジュールは、新しい値と前の値と間の差の絶対値を計算する（１０１４）。 マージテスト・モジュールの比較器１０１６は、タイラー・レジスタ１０１８に格納されている基準値との差を比較し、新しい値と前の値が所定の許容範囲内にあるか否かを示す論理値を生成する。 マージテスト・モジュールから出力された論理値は図３０に示すビット式のＡＮＤブロック１０１０に入力される。 ビット式のＡＮＤの出力は、カラー値、 アルファ値及び深度値の各々が、所定の許容範囲内にあるか否かを示す。許容範囲内にあることを示すのであれば、 上記のように、 画素エンジンは、入力してくる先頭の画素フラグメントをマージする。

【０３７２】

上述のように、 画素フラグメントをマージする方法に対して、多くの可能な変形がある。 他の方法としては、 画素エンジンは、フラグメントリストをサーチして、以下の場合まで画素フラグメントの各々についてマージテストを実行する。１）そのサーチが、リストの末尾に達するまで。又は、２）マージテストを満たす、格納された画素フラグメントを見出すまで。さらに別の方法としては、 画素エンジンは、 例えば、 入力してくる各フラグメントで挿入ソート(insersion sort)を実行することにより画素フラグメントをソートした形式で保持することができる。 画素エンジンは、 入力してくる画素フラグメントを、 視点に最も近接する画素フラグメントで（最も低いｚ値で）のみマージを試みることをできるようにするか、 又はある画素位置に対して格納されている数個の画素フラグメントでマージを試みることをできるようにする。

【０３７３】

画素メモリのオーバフローを防止するための画像領域のサブ分割

幾何形状プリミティブをラスタライズするとき、 タイラーは画素データを画素及びフラグメントバッファ内に格納する。 また、タイラーはこの画素データを処理ステップで解析する。 タイラーはこの画素データを格納するために固定サイズのメモリを使用するので、 メモリ容量が不足する可能性がある。 この問題に取り組むために、 タイラーはメモリ容量を監視して、フラグメントメモリのオーバフローを避けるように、可能なら、その時点でレンダリングされている画像部分のサイズを減少させる。

【０３７４】

一実施例において、 タイラーは、 多くの３２×３２画素チャンクを処理することによ

ってグラフィック出力画像を構築する。 図３２は、 画素及びフラグメントバッファの一部分を図示する図である。 この例に示されるように、 タイラーは関連する５１２個のエントリフラグメントバッファ（１１２０）を使用する３２×３２画素バッファ（１１１８）を解析する。 この具体例においては、 フラグメントバッファは、 ３２×３２画素出力バッファを形成するために後処理ステージで組み合わせられる５１２個の画素フラグメントまで格納することができる。３２×３２出力画素バッファを生成するために５１２個をエントリする画素フラグメントを使用する際に、 微細に碁盤目状配列されたグラフィカルオブジェクト、又はかなりの半透明度を含むオブジェクトをラスタライズするとき、フラグメントメモリを使い尽くす可能性が明確に存在する。 これらの場合、更なるフラグメントメモリは、部分的にカバーされた画素又は半透明性の画素用の画素フラグメントデータを格納する必要がある。 ５１２個の画素エントリを有するフラグメントバッファは、１０２４（３２×３２＝１０２４）個の画素を格納する３２×３２出力バッファと同数の画素の１／２だけを格納する。

【０３７５】

このメモリの限界の問題を緩和するために、 タイラーにおける画素メモリフォーマットは、２レベルの階層分割をサポートするように構成される。 図３３は、この階層分割を示す図である。 フラグメントメモリが32×32画素バッファを処理するときに使い尽くされると、 タイラーは、 画素フラグメントバッファをフラッシュするとともに、 四つの１６×１６画素サブバッファの組（１１２２）に対してプリミティブの入力ストリームを再処理する。 １６×１６画素バッファを５１２個のフラグメント・エントリメモリシステムで処理することは、 所望の出力画素より２倍のフラグメント・エントリを与えることができ、 部分的にカバーされた画素又は半透明性の画素を多く用いて、多岐にわたる場合に対して取り扱うことになる。

【０３７６】

フラグメントメモリが１６×１６画素サブバッファのいずれかを処理するときに使い尽くされると、 タイラーは画素及びフラグメントバッファをフラッシュするとともに四つの８×８画素サブバッファの組に対して、プリミティブの入力ストリームを再処理する（１１２４）。 １６×１６画素のサブバッファの各々は、四つの８×８画素サブバッファに分割されて全部で１６個の８×８サブバッファになる。 ８×８画素バッファを５１２個のフラグメント・エントリメモリシステムで処理することは、 所望の出力画素より８倍の画素エントリを与えることができ、 最も考え得る複合グラフィックオブジェクトを取り扱うことになる。８×８サブバッファの付随的な利点は、８×８画素バッファが、画素データを圧縮するのに使用される圧縮エンジンで必要されるフォーマットとなることであり、更なる画素バッファ分割を圧縮の前に必要としないことである。

【０３７７】

各画素サブバッファ（即ち、 １６×１６又は８×８のもの）は連続的に処理されるときに、 それらの画素は解析されて圧縮エンジンに送出される。 タイラーは完全な３２×３２画素バッファの解析及び圧縮の順で、１６×１６及び８×８サブバッファを処理するので、 全てのサブバッファの処理の完了は、追加処理を要求することなく、 圧縮された形式でシステムメモリに格納された完全な３２×３２画素バッファをもたらす。

【０３７８】

バッファ分割処理は、処理が要求される場合（例えば、 かなりの半透明度、 シャドウ、及び、１つ以上の光源による照明を有する、オーバーラップした微細な碁盤目状のオブジェクト）に対して、回帰的にオンザフライ（on-the-fly）が適用される。この方法を以下に説明する。

【０３７９】

図３４Ａ及び図３４Ｂは、 タイラーにおけるバッファ分割方法を図示するフロー図である。 前処理フェーズでは、ＤＳＰは、レンダリング命令、及び、チャンクと呼ばれる画像領域の間でソートされるポリゴンを含む入力データ・ストリームを生成する。 そして、ＤＳＰは入力データ・ストリームを処理のためにタイラーに送出する。 その入力デ

ータ・ストリーム中のレンダリング命令に応じて、 タイラー内のラスタライザは、入力データ・ストリームの中のポリゴンをラスタライズして画素データを生成する（１１３０，１１３２，１１３６）。

この特別な例において、 本フローチャートは、 ポリゴンがシリアル形式で処理されることを示している。 しかしながら、プリミティブをレンダリングする多くの方法がある。 プリミティブがラスタライズされる手法は、分割処理に対して重要ではない。

【０３８０】

ラスタライザが画素データを生成するとき、 フラグメントバッファの容量を監視する。 この具体例においては、 ラスタライザはフラグメントメモリに加えられる各エントリのバッファカウンタをインクリメントさせ、 画素データを生成するときのカウンタの値をチェックする（１１３８，１１４２）。 バッファカウンタの値が512に達すると、 フラグメントメモリはいっぱいになる。 この点において、 タイラーは現在のチャンク・サイズをチェックして画素データをどのようにサブ分割するかを決定する（１１４４，１１５０）。

ここに記載され図示されている特定の具体例においては、 メモリ分割はフラグメントメモリがその容量、 即ち５１２個の画素フラグメントに達するときに開始される。 しかしながら、 フラグメントメモリが容量いっぱいになる前に復元を開始することもできる。

【０３８１】

チャンク・サイズが３２×３２画素であるときは（１１４４、 タイラーはチャンク・サイズを４つの１６×１６画素チャンクに分割する（１１４６）。そして、 タイラーは画素及びフラグメントのバッファをクリアして（１１４６）、四つの１６×１６サブチャンクに対して、現在のチャンクの入力ストリームをラスタライズすることを開始する（１１５８）。この具体例において、 ＤＳＰは、 チャンクの入力ストリームを再送出する。 サブチャンク間でのポリゴンの再ソートではなく、タイラーは、各サブチャンクに対して、ポリゴンの入力ストリームを繰り返して処理し、 各サブチャンク以外にあるポリゴンを拒否する。 他の方法として、 ＤＳＰは入力データ・ストリームを再処理して、各サブチャンク領域間のストリーム内のポリゴンをソートする。 この他の方法は、 各サブチャンクのポリゴンの数を減少させるが、 ＤＳＰ中の処理負担を増大させる。

【０３８２】

タイラーは、 同様の方法で１６×１６サブチャンクを処理する（１１５０，１１５２）。 現在のチャンク・サイズが１６×１６画素であれば、 タイラーはチャンクを８×８画素に分割して画素及びフラグメントのバッファをクリアする（１１５２）。この具体例において、 タイラーはチャンクを８×８ブロックより小さくにはサブ分割しない。この場合５１２個の要素であるフラグメントメモリの容量は、 画像チャンクを８×８ブロックにサブ分割することによって微細に碁盤目状及び／又は半透明性のオブジェクトを取り扱うのに十分であるべきである。 しかしながら、 ここに記載されているタイラーは１つの可能性がある方法であり、 画像のサイズをサブ分割する必要性は、場面の複雑性、 サポートされたアンチエイリアシング及び半透明性の形成、 フラグメントバッファのメモリ容量のような要因によって変化する。

【０３８３】

バッファカウンタが８×８画素ブロックに対して５１２に達すると、 タイラーは更に８×８画素チャンクと関連する画素フラグメントを解析してバッファスワップを行なう（１１５４）。８×８チャンクが解析された後で、 タイラーはさらに８×８画素チャンクがあるか否かをチェックする（１１５６）。更に８×８画素チャンクがあれば、 続いて次の８×８サブチャンクのポリゴン処理を再び開始する（１１５８）。

他に８×８チャンクが残っていなければ、 タイラーは更に１６×１６画素チャンクがあるか否かをチェックする（１１４８）。 １６×１６画素チャンクが残っていれば、 タイラーは残った１６×１６画素サブチャンクに対してポリゴン処理を再び開始する（１１５８）。 他に１６×１６画素チャンクがなければ、 タイラーは後続するチャンクに対し

て入力データ・ストリームを作り（１１６０）、その中でのポリゴン処理に進む（１１５８）。

【０３８４】

チャンク又はサブチャンクの入力データ・ストリームを処理しているときにフラグメントバッファの容量が超過しないなら、 タイラーは画素及びフラグメントバッファでの画像データの解析に進む（１１３２、１１３４）。 タイラーが現在のチャンクでの入力データ・ストリームの処理を完了していれば、 チャンク又はサブチャンクの解析フェーズを開始する。 例えば、 チャンク・サイズが３２×３２画素であれば（１１６２）、 ３２×３２画素チャンクが解析され、 バッファがスワップされる（１１６４）。 そして、続いて後続するチャンクを取得する（１１６０）（図３４Ａ）。

【０３８５】

チャンク・サイズが１６×１６画素であれば（１１６６）、１６×１６画素チャンクが解析され、 バッファがスワップされる（１１６８）。 タイラーは１６×１６チャンクが更に残っているか否かのチェックに進む（１１４８）。 残っていれば、 次のサブチャンクのポリゴンを再送出することによりポリゴン処理を再び開始し、 残っていなければ、後続するチャンクの入力ストリームをフェッチして、 そのチャンクのポリゴン処理を開始する（１１６０）。

【０３８６】

チャンク・サイズが１６×１６画素でなければ、 それは、 デフォルトによって８×８画素である。 タイラーは次いで８×８画素チャンクを解析し、バッファをスワップする（１１５４）。 そして、 タイラーは残りの８×８サブチャンク、 更に残りの１６×１６サブチャンクを処理する。 残りのサブチャンクの処理を完了した後、 タイラーは後続するチャンクに進む。 処理は、 入力データ・ストリームの中にチャンクが無くなったときに最終的に終了する

【０３８７】

チャンクの処理の間、 データは、 各チャンクが生成する画素フラグメントの最大数を決定するために集められる。 各チャンクを処理した後における５１２個のフラグメントバッファの中でフリーのエントリの数もまた集められる。このデータは、 オブジェクトを再処理するときにバッファ分割がいつ自動的に実行されるべきかを決定するのに使用される。 例えば、 複雑なオブジェクトがゲームの経過中に何回も再描画されているとき、 複雑なオブジェクト処理は、 画素情報の入力ストリームを連続的に再処理するのを避けるために集められた画素バッファデータに基づいてバッファ分割を自動的に行なう。

【０３８８】

１６×１６又は８×８サブバッファへのバッファ分割もまた、 既知の複雑な（即ち、微細な碁盤目状等）画素チャンクがタイラーに送られたときに要求されてもよい。 これは、 画素チャンクがすでに複雑であると知られているときバッファ分割、 画素フラグメントバッファのフラッシュ、 及び入力ストリームの再処理が必要か否かの決定を不要にするとともに、 集中処理を必要とする。

【０３８９】

オーバフローが検出されたときに走査変換処理を再開始する方法として他に少なくとも二つの方法がある。 １つの方法では、 画素エンジンが、 オーバフローが検出されたときに走査変換ブロックが停止するように指示し、 処理されるべきサブチャンク以外の画素位置の画素メモリの全てのフラグメントリストをクリアしてもよい。 これを達成するために、 画素エンジンは、 サブチャンク以外の画素位置の画素バッファにおけるフラグメントリストのポインタを読み取って、 これらの画素位置に関連するフラグメントバッファ内のフラグメントをフリーにすることによりサブチャンク以外でフラグメントリストを参照する。 走査変換ブロックは、 終了したところの現在のチャンクの幾何形状プリミティブの組を続けてラスタライズする。

二つ目の方法では、 走査変換ブロックは、全てのフラグメントメモリをクリアした後で起動する。 この場合、 走査変換ブロックは、起動後チャンクのプリミティブの組の最

初において幾何形状プリミティブのラスタライズを開始する。
オンザフライ・バッファ分割は、極めて複雑な特性（例えば、複数光源、微細な碁盤目形状、半透明性、等）を有するグラフィックオブジェクトを処理するときでも、小さな画素出力バッファ、小容量のフラグメントバッファメモリを使用し、グラフィックオブジェクトの処理の際の、フラグメントデータメモリのオーバフローを低減する手法を提供する。
特定の実施例に関して分割を説明しているが、本発明は種々の他の手法で実行され得ることを理解すべきである。上述したような特定な方法で画像領域を分割する必要はない。むしろ、画像領域は種々のサイズのサブ領域に分割できる。チャンキング・アーキテクチャは、画像のサブ分割には特によく適合するが、フルフレームバッファはより小さい領域に分割されてフラグメントメモリの要求を低減する。メモリ消費を追跡するのに使用される特定のタイプのロジック又はソフトウェアもまた、変形することができる。つまり、本発明の範囲で多くの使用可能な他の方法が含まれる。
画素の後処理
画像処理装置１０６は、画素位置のフラグメントデータを生成した後で、このフラグメントデータをソートかつ解析してその位置でカラーを計算する。上記のように、画像処理装置は部分的にカバーされた画素のフラグメントを生成しかつ保持する。１つ以上のポリゴンのエッジが画素と交差するとき、又はポリゴンが半透明性のとき、画素はポリゴンにより部分的にカバーされる。フラグメントデータを保持してアンチエイリアシングと半透明処理を実行するには十分なメモリ容量を必要とする。レンダリングされるポリゴンの数が増加すると、画素データ及びフラグメントを格納するメモリの容量もまた増加する。
増加したメモリ要求に加えて、フラグメントを解析するのに要求される処理量も多くなる。Ｚバッファへのアプローチにおいて、フラグメントデータは深度値でソートされる。一般的に、プリミティブのデータは、レンダリングのために到着する深度の順にソートされない。プリミティブのデータが任意の深度の順で到着するので、画像処理装置はフラグメントデータを生成した後でフラグメントデータをソートしなければならない。ソートされたデータは、画素位置でカラー値と、可能であれば、アルファ値を決定するように処理される。各画素位置での様々なフラグメントが、そのカラー値に対して寄与することになる。アルファ値が計算されると、フラグメントの数及び処理の複雑さもまた増加する。
上記のように強調された理由によって、進歩的なアンチエイリアシング及び半透明処理をサポートするためのメモリ及び処理要求は不可欠である。一方では、複雑なアンチエイリアシング及び半透明処理の計算をサポートすることと、他方では、メモリ要求を低減することの間には矛盾がある。システムのコストを低減するために、メモリの使用を最小限にすべきだが、進歩的なアンチエイリアシング及び半透明処理は、通常、更なるメモリを必要とする。これらの、進歩的な構成をメモリの要求を最小にしつつリアルタイムのシステムで保持することは困難でさえある。
一実施例において、本発明に係るシステムであれば、一度に１つのチャンクのプリミティブをレンダリングし、その結果メモリを減らして後処理ステップでフラグメントの解析を可能とする。画素データが１つのチャンクで生成する間に、他のチャンクの画素データを解析することができる。フラグメントのソート及び画素の解析に影響を与える数多くの利益がチャンキングの概念からもたらされる。画像処理装置がチャンク内の画素を解析した後ではラスタライズ処理の間に生成した多くのデータは保持されるべきではないので、メモリの要求は十分に減らされる。本画像処理装置は、チャンクを解析した後で解析されたカラー部分のみを保持すればよい。
シリアル処理でチャンクをレンダリングすることの他の利点は、画素フラグメントメモリがメモリアクセスのオーバーヘッドを低減するのに実現できることである。典型的なグラフィックシステムは、外部メモリを使用してカラー、深度及びフラグメントバッファを設ける。この外部メモリを有機的に構成してリアルタイム画像処理の厳密なバン

ド幅の要求を満たすことはかなり困難である。３２×３２画素領域のようなチャンクのレンダリングをサポートするのに必要な画素フラグメントメモリは外部メモリに設けられるべきではなく、 それに代えて、 ラスタライズ及びアンチエイリアシング機能を実行する同一のハードウェア上に設けることができる。 例えば、 上記の具体例において、 フラグメント及び画素バッファは単一の集積回路チップ上に設けることができる。

チップメモリを使用すると外部メモリに関連するバンド幅の問題を簡素化することができる。 チップメモリは多段メモリを効率的に使用することができる。 例えば、 １段を画素バッファに使用し、 他段をフラグメントの記録に使用することができる。

チップメモリの他の利点は、 多ポートメモリの装備を低価格かつ容易に行なうことができることである。 画素及びフラグメントバッファの性能は、 多ポートメモリを使用することにより増大することができ、 その結果同時読み込み及び／又は書き込みが画素処理速度当たり１クロック(one clock per pixel processing rate) を達成することができる。 チャンクが別々にレンダリングされるときはフラグメントバッファがかなり小さいので、 それはチップ上に装備される。 メモリの小サイズ及びチップ上のメモリの存在はともに、 多ポートメモリの使用を実現可能にし、かつコスト的にも効果的である。 一方、 外部多ポートメモリは、ビット当たりのコストは高く、チップ間の接続に起因して高価である。

チャンキングに関する他の重要な利点は、 フレームの一部の画素を、 他の部分の画素が解析されている間に生成することができることである。 このように、 本アプローチは、 全部のフレームの画素を生成してそれらの画素を解析するのではなく、 画素の生成とともに解析をオーバーラップさせることができ、 その結果システムの伝達遅延を低減することができる。

本システムの一実施例において、 画像処理装置は後処理ステップでフラグメントを解析する。 画素エンジンが画像の一部の画素データを生成している間に、 アンチエイリアシング・エンジンが画像の他の部分のフラグメントを解析する。 上記のように、 画素データはダブルバッファされている。 即ち、 アンチエイリアシング・エンジンが一方のバッファをアクセスしている間に画素エンジンは他方のバッファをアクセスすることができる。 画素エンジンがチャンクの画素を生成した後で、 タイラーはバッファスワップを実行する。 画素エンジンは後続するチャンクの画素を生成するとともに、 アンチエイリアシング・エンジンは前のチャンクの画素を解析する。

【０３９０】

画素データをダブルバッファすることもできるが、 好ましい実施例においては、フラグメントバッファに二つのポートが設け、 そのバッファを画素エンジン及びアンチエイリアシング・エンジンが同時にアクセスしてもよい。 画素エンジンは一方のポートを介してフラグメントデータをフラグメントバッファに書き込み、アンチエイリアシング・エンジンは他方のポートを介してフラグメントデータを書き込む。

【０３９１】

本実施例において、 ダブルバッファされるとともに二つのポートが設けられたメモリシステムにより、 画像処理装置は画像データの生成及び画素の解像をオーバーラップさせることができる。

画素処理装置は解析処理を完了する前に深度順にフラグメントデータをソートする。 一般的に、 画像処理装置は画素を生成するとき、 またレンダリングされるべき画像の部分の画素を生成した後で画素データをソートすることができる。 例えば、 画素エンジンは、 フラグメントデータをフラグメントバッファに書き込むときに挿入ソートを実行することができる。 加えて、 画素エンジンは、 画像の全て又は一部の画素データを生成するのを完了した後でフラグメントデータをソートすることができる。 画素エンジンはまた、 入力してくる画素データを拒否する場合は、 フラグメントをソートすることができる。 画素エンジンは、 入力してくる画素データを拒否するときは、 フラグメントバッファに書き込みを行なうべきではないので、 入力してくる次の画素が到来する前にフラグメントのソートを実行することができる。 ここで、後者のアプローチをフラグメン

トの「背景ソーティング」とも称する。

【０３９２】

挿入ソートは、 入力してくるフラグメントをフラグメントバッファの他のフラグメントに深度に関してソートすることに関係している。 潜在的に挿入ソートは画像データを生成する処理を減速するから、 潜在的にリアルタイムシステムには好ましくない。 入力してくるフラグメントの適切な挿入ポイントを見つけ出すためにフラグメントバッファを探すと、 望ましくないオーバーヘッドを生じさせ、 加えて、 ハードウェアによる実現に関して、 追加のハードウェアを必要とするとともに画素エンジンの設計を複雑にする。

【０３９３】

挿入ソートの他の方法として、 画像処理装置が画像の一部の画素の生成を完了した後でフラグメントをソートしてもよい。 幾つかのシステムは一旦画像データの全フレームをレンダリングする。 そのようなシステムにおいて、 ビュー空間で各画素位置毎にフラグメントをソートすると、 特に、 リアルタイムシステムにおいて、 実質的な処理時間を必要とするとともに望ましくない遅延を生ずる。 ソートに必要な時間は画素当たりのフラグメントの数に応じて、 かつすでに実行されている挿入ソートの程度に応じて変わるため、 ソート動作は他の画素動作が生じるのを妨げ、 性能を低下させる。

【０３９４】

ビュー空間 の一部を一度にレンダリングすることにより、 画像の一部のフラグメントを、 次の部分をラスタライズしながら、 ソートすることができる。 本質的に、 アンチエイリアシング・エンジンは、 後続するチャンクのフラグメントを生成しているときに１つのチャンクのフラグメントをソートする。

画素の生成及び解析が上記のようにオーバーラップしていたとしても、 画素エンジンが画像の部分の画素を生成するときに画像の一部のフラグメントのソートを、 実行することは有益である。 画素フラグメントを背景ソーティングすると、 画素エンジンが一組のプリミティブの画素生成を完了した後におけるフラグメントのソーティングのオーバーヘッドを低減させる。

【０３９５】

一実施例において、 背景ソーティングは同時に実行され、 画素動作がフラグメントのソートに必要とされるレイテンシを低減、場合によっては無くす。その設計は、多くの画素が部分的にカバーされ、 それ故フラグメントバッファを使用できなくするという効果もたらす。 背景ソーティングは、 この予備のバンド幅を使用してフラグメントバッファの一組のフラグメントのソートを実行する。

【０３９６】

ソートの後で、 画像処理装置は、画素位置のフラグメントを解析してその画素位置のカラーを決定する。 アルファが考慮されていなければ、 画像処理装置は画素位置の深度がソートされたリストにおけるフラグメントのカラー及びカバレッジデータに基づいてカラーの累積値を計算する。 アルファがカバレッジデータに加えて考慮されるなら、 画像処理装置は、 画素位置における深度がソートされたリストのフラグメントのカラー、 カバレッジ、 及びアルファに基づいてカラーの累積値を計算する。

【０３９７】

一般的に、 画像処理装置は、 ビュー空間全体、 又はその時のビュー空間の一部のみに対応する画素位置のフラグメントを解析する。 上記の実施例において、 画像処理装置は、 チャンクと呼ばれるビュー空間の一部の画素位置を解析する。 フラグメントの解析は、 フラグメントが生成するとともにソートされた後で生じる。

【０３９８】

フラグメントの解析は、 画素の全てのフラグメントが単一のカラー値とアルファ値を計算するのに組み合わせる間における処理である。 この単一のカラーとアルファはカラーバッファの中に書き込まれる（次いで圧縮され、 ｇスプライトに保持される）。

解析されたカラー値の計算は、カバレッジ情報を計算かつ維持して、次に続く層をスケ

ーリングするために、正しくスケーリングされたカラーコントリビューションを各層から累積することを含む。 この累積は、 前部から後部、 又は、後部から前部での深度の順に実行することができる。前部から後部へのアプローチでは、後部から前部へとは反対に、 空間カバレッジのデータは、続く層のカバレッジを決定するのに使用される。 カバレッジの場合とは異なり、 アルファデータは、全ての画素領域に対して等しく適用される。

【０３９９】

前部から後部に関するものは、 ソートされたフラグメントの記録に対するカラーとアルファを計算する式は以下のようになる。

アルファ(Alpha) は最大値（アルファ値を反転）に初期化され、 カラー(Color) は０に初期化される。

```
Anew=Aold-Aold*Ain);
Cnew=Cold+(Cin*(Aold*Ain));
```

後部から前部に関するものは、ソートされたフラグメントの記録に対するカラーとアルファを計算する式は以下のようになる。

アルファ及びカラーは０に初期化される。

```
Anew=Ain+((1-Ain)*Aold);
Cnew=(Cin*Ain)+((1-Ain)*Cold);
```

ハードウェアによる実現例では、 解析処理はハードウェア依存度が少ないので前部から後部への順が好ましい。

フラグメントを深度、 カラー、 カバレッジのみで（アルファなしで）フラグメントを累積する擬似コードの例は以下のように記述される。

NUM __CVG __BITSはカバレッジマスクのビットの数であり、 MAX __ALPHA はアルファの最大値である。

```
for (each fragmented pixel location) {
   ColorAccum = 0;
   CoverageAccum = 0;
   while (fragment list is not empty(フラグメントリストは空ではない)) {
       scan fragment list and extract closest fragment(coverage, color);
       (フラグメントリストを走査し、 かつ最も近いフラグメントをレンダリングする)
       ColorScale=CountSetbits(coverage&((CoverageAccum))/NUM__CVG __BITS;
       ColorAccum+=ColorScale* color;
       ColorAccum =coverage;
   )
   ColorAccum is pixel color
   (ColorAccumは画素のカラーである)
)
```

【０４００】

フラグメントを深度、カラー、 カバレッジ、及びアルファによって累積するには、アルファ値を計算して各サブサンプルのために保持する必要がある。これは、 各フラグメントに対するカバレッジマスクとアルファ値の組み合わせによる。一般的に、 累積中のいずれかの層で累積されたアルファが以前の層のアルファ値の全ての機能であると言える。 カバレッジマスクによって、各サブサンプルは「前の」アルファ値の異なる組を潜在的に有する。 なぜならば、 カバレッジビットが明確である層はそのサブサンプルに寄与していないためである。

【０４０１】

アルファとカバレッジによってフラグメントを解析する１つのアプローチはある層の各サブ画素のカラーを計算し、 サブ画素位置からのコントリビューションを加えてトータルカラーコントリビューションを決定することである。 このアルファのスケール値に、

サブ画素のカラーを乗算して、サブ画素のカラーコントリビューションを決定する。 ある層のカラーはサブ画素からのカラーコントリビューションを合計することにより決定される。

【０４０２】

サブ画素のカラー及びアルファを個別に累積する一例は以下の通りである。

```
for（each fragmented pixel location） {
  ColorAccum=0;
  AlphaAccum[NUM__CVG __BITS]= {MAX __ALPHA,MAX __ALPHA,...MAX__ALPHA } ;
  while（fragment list is not empty） {
      scan fragment list and extract closest fragment(coverage, color, alpha);
      for(i=0;i<NUM __CVG __BITS;i++) {
        //  このビットがカバレッジのマスクに設定される場合、
        if(coverage>>I)& 0x1 {
          //  アルファのスケール値－このカラー用コントリビューションを計算。
          AlphaScale=(Alpha*AlphaAccum[i]);
          //  アルファによりスケーリングしたカラーを加える。
          ColorAccum +=(color*AlphaScale)*(1/NUM __CVG __BITS)} ;
          //  サブサンプルの累積されたアルファを計算する。
          //  AlphaAccum=AlphaAccum*(MAX__ALPHA-alpha)=
          //  AlphaAccum-AlphaAccum*alpha
              AlphaAccum[i]=AlphaScale;
        }
      }
  }
  ColorAccum is pixel color
}
```

【０４０３】

四つのサブ画素位置を使用する例は、フラグメントの解析を説明するのに都合がよい。 本例においては、 それぞれがカバレッジマスク、 アルファ及びカラー値を有する三つのフラグメントを考える。 最初の状態は下表に示されている。 本例においては、 前部から後部へのアプローチを使用するカラー及びアルファを累積する。最初のアルファは十分に半透明を意味する１に設定される。 各層のデータは次の通りである。 フラグメント０、 アルファ＝０．５、 カバレッジマスク（ｃｍ）＝００１１、 及びカラー＝Ｃ０；フラグメント１、 アルファ＝０．３、 ｃｍ＝１０００、及びカラー＝Ｃ１；フラグメント２、 アルファ＝０．８、 ｃｍ＝０１０１、 及びカラー＝Ｃ２。各フラグメントのデータは下表で与えられている。

アルファ値は１に初期化され、 アルファ・カバレッジのアレイは下記のように示される。

【０４０４】

【表４】

1	1
1	1

【０４０５】

カラーを計算するために、 各サブ画素位置のカラー値には新しいアルファ及びカバレッジのアレイからのアルファ値が乗算される。 サブ画素位置の結果は四つ（サブ画素位置の数によって除算されたもの）に除算される。 最後に、 サブ画素位置の全てからのコントリビューションは、累積されたカラーを算出するために合計される。

【０４０６】

【表５】

カバレッジマスク	カラー	新たなフラグメントのアルファ	カバレッジアレイのアルファからのアルファ	サブ画素のコントリビューション
1	C_0	0.5	1	1/4
1	C_0	0.5	1	1/4
0	C_0	0.5	1	1/4
0	C_0	0.5	1	1/4

【０４０７】

公式Alpha'=Alpha*(Max __alpha-new __alpha)を使用して、 画像処理装置は新しいアルファを各画素位置に対して個別に計算し、それを下表のアルファ・カバレッジのアレイに格納する。

【０４０８】

【表６】

0.5	0.5
1	1

【０４０９】

フラグメント１のコントリビューションは下表に記載されている。

【０４１０】

【表７】

カバレッジマスク	カラー	新たなフラグメントのアルファ	カバレッジアレイのアルファからのアルファ	サブ画素のコントリビューション
0	C_1	0.3	0.5	1/4
0	C_1	0.3	0.5	1/4
0	C_1	0.3	1	1/4
1	C_1	0.3	1	1/4

【０４１１】

新しいアルファ・カバレッジのアレイは次の通りである。

【０４１２】

【表８】

0.5	0.5
0.7	1

【０４１３】

フラグメント２のコントリビューションは下表に記載されている。

【０４１４】

【表９】

カバレッジマスク	カラー	新たなフラグメントのアルファ	カバレッジアレイのアルファからのアルファ	サブ画素のコントリビューション
1	C_2	0.8	0.5	1/4
0	C_2	0.8	0.5	1/4
1	C_2	0.8	1	1/4
0	C_2	0.8	0.7	1/4

【０４１５】

フラグメント２の後の、フラグメントのアルファ・カバレッジのアレイは次の通りである。

【０４１６】

【表１０】

0.5	0.1
0.7	0.2

【０４１７】

この方法は、アルファの計算及びカラーコントリビューションのフラグメント毎の2*NUM__CVG__BITSの乗算（４×４の場合、２＊１６＝４８）を必要とする。カバレッジのマスクにおけるビット数が２＊＊ｎのサイズ（典型的な場合）とすると、(1/NUM__CVG__BITS) のスケーリングはシフトによって為される。

図３５は、４×４のサブ画素領域（１２２４）に分割された画素に対して上記のアプローチのハードウェアによる実現例を示す概略図である。解析ハードウェアはアルファ・カラーアキュムレータ（ＡＣＡ）（１２２６）と呼ばれる１６個の同じ処理記憶ユニットの組を含み、各ＡＣＡは画素のうちの１つのサブ画素領域専用である。各画素位置のフラグメントリストの処理の間、各フラグメントのカバレッジマスクは解析ハードウェアの処理マスクとして使用される。ＡＣＡは、アルファのスケール値、カラーの累積値、及びアルファの累積値の乗算を実行する。(1/NUM__CVG__BITS) のスケーリングは、上記のようにシフトによって実行される。一旦全てのフラグメントが所定の画素位置に対して処理されると、出力部は階層上に１６個のサブ画素の全てをカラー及びアルファ値を組み合わせる（１２２８）。出力用の処理装置は入力してくる二つの値を組み合わせて、２で割る。ハードウェアをパイプラインして、画素解析処理はフラグメントのエントリ毎の、単一のハードウェアのブロックのみを使用する。

【０４１８】

別の技術は、同様に各層で累積された同じアルファを有するサブ画素を処理することによってハードウェア要求を低減する。この技術は、サブサンプルが累積された固有のアルファ値を有する状態が徐々に生じるということに基づいている。最初は、サブサンプルのアルファの全ては０（透明）に設定されている。最初のフラグメントの累積は、多くとも１つの固有のアルファ値を加えることができ、最初のアルファ値を保持しているサブサンプルの１グループと同じ新しいアルファ値を有する他のグループをもたらす。次のフラグメント累積は、たった四つの固有のアルファ値をもたらす。全体としては、「ｎ」個のフラグメント累積の後で、可能性がある固有のサブサンプルのアルファ値の数は、２＊＊ｎ（又は、さらに正確には、MIN(2**n(NUM__CVG__BITS))となる。

【０４１９】

この別の技術は、サブサンプル毎ではなく、サブサンプル内の各固有のアルファ値用のカラー値のスケーリング及び累積を実行することによって、要求される累積値の数を低減させる特徴を使用している。この技術によって、せいぜい１つの累積が、第１のフラ

グメントとして、 第２のフラグメントとして２つ、 第３のフラグメントとして３つ、等、 画素のサブサンプルの数まで生じる（例えば、 ４×４のサブサンプルのアレイによって、最悪の場合はフラグメント毎に１６個の累積が生じる）。

【０４２０】

この技術は、 フラグメントの累積の間固有のアルファ値の組とそれらの関連するカバレッジマスクを維持することを基礎としている。 その内容は最小数のカラー累積値を累積する。

アルファ及びカバレッジマスクは、 これらのエントリのあるサブセットがある時間で実際に効力がある（又は「使用中」）NUM __CVG __BITS要素のアレイに格納される。「使用中」のエントリは固有のアルファ値の現在のセットを保持するものである。 使用中のエントリビットインデックスにおけるアレイ要素が使用中であることがビットセットで示されるNUM __CVG __BITSビットのマスクによって識別される。 {固有のアルファ、 カバレッジのアレイ} の対におけるカバレッジのアレイ中の第１のビットセットはその対が格納されているアレイ要素を規定するという慣例が使用されている。 アレイを３つのフラグメントの累積によって（３つのサブサンプルを使用して）初期化されるとともに更新する手法として次の例を挙げる。

【０４２１】

初期状態（Ｘは「注意不要」値を意味する）:

```
        0b0001  //使用中のマスク
         {1., 0b1111} //アルファとカバレッジの対
         {X, 0bXXXX }
         {X, 0bXXXX }
         {X, 0bXXXX }
累積フラグメント {.5alpha*/.0b0011カバレッジマスク*/}
        0b0101  //使用中のマスク
         {.5, 0b0011} //アルファとカバレッジの対
         {X, 0bXXXX }
         {1., 0b1100}
         {X, 0bXXXX }
累積フラグメント {.3, 0b1000}
        0b1101  //使用中のマスク
         {.5, 0b0011}  // アルファとカバレッジの対
         {X, 0bXXXX }
         {1., 0b0100}
         {.7, 0b1000}
累積フラグメント   {.8, 0b0101}
        0b1111  //使用中のマスク
         {.1, 0b0001} //アルファとカバレッジの対
         {.5, 0b0010}
         {.2, 0b0100}
         {.7, 0b1000}
```

最初のアルファ・カバレッジのアレイは以下のように記載される。

【０４２２】

【表１１】

x	1
x	x

【０４２３】

使用中のマスクは、 アレイのマスクに格納されている位置を特定する０００１である

。 対応するアレイのマスクは以下の通りである。

【０４２４】

【表１２】

xxxx	1111
xxxx	xxxx

【０４２５】

フラグメント０の後で、 アルファ・カバレッジのマスクは以下のように表される。

【０４２６】

【表１３】

x	0.5
x	1

【０４２７】

使用中のマスクは０１０１であり、 アレイのマスクは以下の通りである。

【０４２８】

【表１４】

xxxx	0011
xxxx	1100

【０４２９】

設定された使用中のマスクの各要素では、 アレイのマスクは新しいフラグメントに対してカバレッジマスクとＡＮＤがとられ、 アルファ値に変化があるか否かを判別する。新しいアルファがあれば、アレイのマスクの新しい値は、 （アレイのマスクＡＮＤ ＮＯＴカバレッジマスク）で計算される。アレイのマスクに新しい値があれば、それを適切な位置に格納する。

フラグメント１の後で、アルファ、カバレッジマスクは以下のように表される。

【０４３０】

【表１５】

x	0.5
0.7	1

【０４３１】

使用中のマスクは１１０１であり、アレイのマスクは以下の通りである。

【０４３２】

【表１６】

xxxx	0011
1000	0100

【０４３３】

フラグメント２の後で、アルファ・カバレッジのマスクは以下のように表される。

【０４３４】

【表１７】

0.5	0.1
0.7	0.2

【０４３５】
使用中のマスクは１１１１であり、アレイのマスクは以下の通りである。
【０４３６】
【表１８】

0010	0001
1000	0100

【０４３７】
ある時点における固有のアルファ値の数は使用中のマスクのセットビットの数に等しい。 完全解は二つのステップを含む。 第１のステップは、 １つの累積がカバレッジ／アルファのアレイにおける「使用中」エントリ毎に要求される必要なカラー累積を実行することである。 第２のステップは新しいフラグメントの値とともにカバレッジ／アルファのアレイを更新することである。
【０４３８】
この技術の（4×4サブサンプルに対する）完全な実現例は以下の通りである。

```
for (each fragmented pixel location) {
        //      初期状態( 画素毎)
        InUsemask = 0x0001;
        CoverageArraymask[16] = {0xffff, 0, ...,0} ;
        CoverageArryAlpha[16] = {MAX __ALPHA.MAX __ALPHA. ....,MAX __ALPHA } ;
        ColorAccum = 0;
        while (fragment list is not empty) {
        scan fragment list and extract closest fragment(coverage, color, alpha);
        //  このフラグメントのカラーを使用中の各要素のColorAccumに累積する。
        InUseMaskScratch = InUseMask;
        while(InUseMaskScrach ! = 0x000) {
            //  使用中のマスク消去時に第１のビットセットを見つける。
            Index = FindFirstSetBit(InUseMaskScratch);
            //  マスク消去時にこのビットをクリアする。
            InUseMaskscratch & = ((0x1 << Index);
            //  このエントリの旧（現）アルファを読み込む－これは、 カバー
            //  されていない領域を更新する（新しく「使用中」になる）。
            AlphaOld = CoverageArryAlpha[Index];
            //  アルファのスケーリング係数の累積のために
            //  カラーをスケーリングし、続く層のアルファを計算するのに使用。
            AlphaScale = AlphaOld*alpha;
            //  次の層のアルファを計算する－これをアルファのアレイを更
            //  新するのに使用する
            AlphaNext = AlphaOld*(MAX __ALPHA-alpha) = AlphaOld-AlphaOld*alpha
            AlphaNex = AlphaOld-AlphaScale;
            //  オーバーラップしたカバレッジのマスクを計算する－これは、
            //  新しいフラグメントによってカバーされるアレイのエント
            //  リの一部であり、 カラーを累積して新しいアルファ値でアレイ
```

```
            //  を更新する。
            AccumCvgMask= Coverage& CoverageArrayMask[Index];
            if(AccumCvgMask ! = 0x0000)[
              //  カラーを累積する。
              nCoverageBits = CountSetBits(AccumCvgMask);
              ColorAccum+ = color*(AlphaScale * nCverageBits/NUM__CVG __BITS)];
              //  カバーされた部分のアルファを更新する（これは「新しい」
              //  使用中の要素をもたらすか、 又は古いものを上書き
              //  する）。
              Index2 = FindFirstSetBit(AccumCvgMask);
              InUseMask ? = (0x1 << Index2);
              CoverageArrayMask[Index2] - AccumCvgMask;
              CoverageArrayAlpha[Index2] = AlphaNext;
             )
             //  カバーされていない領域のマスクを計算する－これは、 新し
             //  いフラグメントによって不明瞭でないアレイのエントリの部
             //  分であり、 カバレッジを更新する（アルファは同じ値をとる）。
             IndateCvgMask = (Coverage & CoverageArrayMask[Index];
             if(UpdateCvgMask! = 0x0000) {
                Index2 = FindFirstSetBit(UpdateCvgMask);
                InUseMAsk? = (0x1 << Index2);
                //      カバーされていない部分のアルファを更新する－これ
                //      は「新しい」使用中の要素をもたらすか、 又は古いも
                //      のを上書きする（このようにそれが新しい場合は、
                //      アルファ値をコピーする）。
                CoverageArrayMask[Index2] = UpdateCvgMask;
                CoverageArrayAlpha[Index2 ] = AlphaOld;
            }
        )
    )
    ColorAccum is pixel color
}
```

【０４３９】

この算術計算の中心は、 固有のアルファ値毎に全部で３つの乗算を必要とするカラー累積である。

```
ColorAccum+=color*(alpha*AlphaOld*(nCoverageBits/NUM__CVG __BITS));
```

第３の乗算はサブサンプルの数によっては若干簡素化される場合があることに注意されたい。 １６個のサンプルに対して、 第３の乗算は０．４の固定小数点を含み、 この乗数は８×４とできる（ここで、 他の乗数は８×８であろう）。また、２＊＊ｎのサイズのカバレッジマスクに対して、 上述の除算は単にシフトさせるのみであることに注意されたい。

この技術は最低値の場合、下式による全ての累積を必要とする。

【０４４０】

【数１１】

$$\sum_{n=1}^{NumFrags} MIN(2^n,16)$$

【０４４１】

典型的な場合は上記の値を大きく下回る。最低値の場合は、新しいフラグメントのカバレッジが各「使用中」のアレイ要素に設定値及び未設定値の双方を有するときのみ生じるからである。

１つの有効な最適化は十分に不透明性のアルファ値を有するカバレッジマスクの位置を追跡することである。これは、フラグメントが非透明性でない半透明度の値によらずに幾何学的カバレッジの一部によって生成されている場合に有効である。これらのフラグメントは、通常、十分に不透明性の半透明値を有する。この最適化は、追加的なマスク値、即ち OpaqueAlphaMaskを保持することにより実行される。この OpaqueAlphaMaskはアルファが完全に不透明性であるフラグメントのカバレッジマスクにおいてＯ－リング (O-Ring) によって設定される（これは、フラグメントのコントリビューションを累積した後で実施される）。対応するサブサンプルに対して更なるカラーコントリビューションが生じ得ないので、このマスクは次のフラグメントのマスクでは、ビットを無視するように使用される。

【０４４２】

別の可能な最適化は、同一のアルファ値を有する位置を統合することである。しかし、これを実現するにはかなり処理負担が大きくなり、０でもなくMAX__ALPHA でもない同じアルファ値が発生することは、ほとんどない。

上記した例及び擬似コードは前部から後部への深度のソートを使用する。同じ計算を深度に関して後部から前部へソートすることは同じように可能である。また、上記の計算は、アルファ成分が予め乗算されていないカラー成分を使用する。同じ技術を、若干異なる算術計算（及び同じ制御フロー）によって、予め乗算されたカラー成分に適用する。

【０４４３】

図３６は、アンチエイリアシング・エンジンにおけるハードウェアの最適化されたフラグメント解析サブシステムの具体例を図示するブロック図である。このサブシステムへの入力は、深度に関してソートされたフラグメントの記録のフローである。ここに示されているように、フラグメントの記録はＲＧＢカラー値、アルファ値、及びカバレッジマスクを含む。この特別なフラグメント解析サブシステムは、前部から後部への順にフラグメントの記録処理を行なうとともに、各フラグメントの層を処理するときに画素位置のカラー値を累積する。このサブシステムは、共通のアルファを有する固有の画素領域を追跡し続けるのでカラー値を累積するのに必要とされるハードウェアを最小にする。これは、フラグメント解析サブシステムが、各サブ画素領域に対して個別に行なうというより、各固有の画素領域に対して一度にスケーリングし、且つカラー値を累積することを可能とする。

【０４４４】

上記の疑似コードに関して説明したように、フラグメント解析システムは、フラグメント記録のリストを解析する前に、使用中のマスク１２３６、カバレッジマスクのアレイ１２３０、累積されたアルファ値のアレイ１２３０を初期化する。使用中のマスク１２３６の要素は、共通の累積されたアルファを有する１つ以上のサブ画素領域をそれぞれ含む、複数の画素領域を表わす。カバレッジマスクは、画素領域によってカバーされるサブ画素位置を与える。累積されたアルファのアレイは共通のアルファを有し、かつ対応する画素領域の累積された固有のアルファ値を保持する。 この特定のカバレッジのアレイ１２３０は累積されたアルファ値及びカバレッジマスクを格納する。

【０４４５】

使用中のマスク、 カバレッジアレイのマスク、 及びカバレッジアレイのアルファを初期化した後で、サブシステムはフラグメントの記録を開始し、視点に最も近いフラグメントの記録を始める。タイラー上のアンチエイリアシング・エンジン４１２の１つの実現例において、走査変換ブロック３９５及びテクスチャフィルタ・エンジン４０１がチャンクのラステライズを完了した後、アンチエイリアシング・エンジンは後処理ステージでフラグメントリストをソートする。アンチエイリアシング・エンジンはフラグメントリストの各フラグメントの読み込みを先頭から始めて、かつインデックス及び深度に関してソートされたアレイの中へのエントリを所々で行なう。このアレイ中の各インデックスは、リス

ト中の画素フラグメントのＲＧＢ、アルファ及びカバレッジデータを格納するフラグメントバッファ位置を示している。画素フラグメントの読み込みの際に、アンチエイリアシング・エンジンは、 アレイのエントリが画素のフラグメント及び対応する深度値に対して、深度値でソートしたインデックスのアレイからなるように、挿入ソートを実行する。一旦リストがソートされると、フラグメント解析サブシステムは、ソートされたアレイの中の各エントリをこれらのエントリがアレイの中に格納されている順に読み込むことによって深度に関してソートされたフラグメントを検索する。これは、フラグメント解析サブシステムが深度に関してソートされる順にリスト中の画素フラグメントのＲＧＢカラー値、アルファ及びカバレッジマスクを検索することができる。

【０４４６】

リスト中の各フラグメントの記録を処理するときに、サブシステムは共通のアルファを有する画素領域を追跡する。サブシステムは、リスト中のフラグメントの記録が共通のアルファを有する各画素領域にオーバーラップするか否かを判別する。オーバーラップする場合、サブシステムは、現在のフラグメントとオーバーラップする現在の画素領域の部分に対して、累積されたカラーを計算する。現在の画素領域にオーバーラップがあると、サブシステムはまた、このオーバーラップによって生じる１つ以上の新しい画素領域を決定して、それらを追跡する。

【０４４７】

現在のフラグメント（１２３２）に対して、サブシステムは、使用中のマスクの各要素を介してループする。カバレッジアレイループ制御装置1234は、使用中のマスク（１２３６）を維持するとともに、各フラグメントの記録を処理する際に必要になる更新を行なう。使用中のマスクのエントリを介してループするとき、カバレッジアレイループ制御装置は、新規なカバレッジ制御装置１２３８と接続してその動作を制御する。新規なカバレッジ制御装置１２３８は、現在のフラグメントが現在の画素領域にオーバーラップするときに必要になるカバレッジアレイのマスク及びアルファ１２３０の更新を行なう。

【０４４８】

新規なカバレッジ制御装置１２３８は、使用中のマスク内の現在のエントリに関連するカバレッジアレイのアルファから、格納且つ累積されたアルファ（Aold）を読み込み、カラーをスケーリングするように使用するとともに、次のフラグメントの層Anext（１－A＊Aold）のアルファを計算するように使用されるアルファのスケーリング係数（A＊Aold）を計算する。この新規なカバレッジ制御装置１２３８は、アルファのスケーリング係数（A＊Aold）を、現在のフラグメントのカラーデータをスケーリングするのに使用される、スケール及び累積制御装置1246に送出する。新規なカバレッジ制御装置１２３８はまた、次のフラグメントの層、Anext（１－A＊Aold）のアルファを計算して、それをカバレッジアレイ１２３０内に、対応するカバレッジアレイのマスクに従って、格納する。

【０４４９】

共通の累積されたアルファを有する各画素領域に対して、フラグメント解析サブシステムは、現在のフラグメントがフラグメント及び画素領域のカバレッジマスクの挿入を見つけることにより現在の画素領域をオーバーラップするか否かを決定する。

現在のフラグメントが現在の画素領域とオーバーラップするなら、サブシステムは、１）画素領域のオーバーラップ部分の累積したカラーを計算し、かつ２）使用中のマスクと、この使用中のマスク要素に対応するカバレッジアレイのマスク及びアルファ（カバレッジアレイのアルファ）を更新する。

スケール及び累積制御装置１２４６は、現在のフラグメントによってカバーされた固有の画素領域の各々に対して、累積されたカラーを計算する。このスケール及び累積制御装置は、カバレッジスケーラ（scaler）１２４０、カラースケーラ１２４２、及びカラーアキュムレータ１２４４を有する。カバレッジスケーラ１２４０は、カバレッジのスケーリング係数（現在のフラグメント／全サブ画素位置でオーバーラップする現在の画素領域のサブ画素位置の数＊A＊Aold）を計算する。カラースケーラ１２４２は、現在のフラグメント（１２３２）のカラー値（ＲＧＢ）を読み込むとともに、それらにカバレッジスケー

ラ１２４０からのカバレッジのスケーリング係数を乗算する。最後に、カラーアキュムレータ１２４４は更新され、且つ、累積されたカラー値を計算するために、スケーリングされたカラーに対して累積されたカラーを加える。

【０４５０】

現在のフラグメントが現在の画素領域にオーバーラップするとき、カバレッジアレイループ制御装置１２３４は新しい画素領域に対応してエントリを含むように使用中のマスク１２３６を更新する。これは、単に現状の使用中のマスク要素を上書きするか、新しいものを生成することによって行ってもよい。カバレッジループ制御装置はまた、カバレッジアレイのマスク１２３０を新しい画素領域のカバレッジに更新するとともにこの新しい画素領域の累積されたアルファを設定するように新規なカバレッジ制御装置に指令する。新規なカバレッジ制御装置１２３８は新しい画素領域に対応する新しいアルファのカバレッジアレイのエントリをAnextに設定する。

現在のフラグメントのみが画素領域の一部を（その全てをオーバーラップするというよりむしろ）カバーするとき、新規なカバレッジ制御装置１２３８は二つの新しい画素領域、即ち、１）現在のフラグメントがオーバーラップする画素領域の一部と、２）現在のフラグメントによって不透明でなくなる画素領域の一部を生成する。この場合、サブシステムは、不透明でない部分のカバレッジを計算するとともに、それに対してアルファを設定して、元の画素領域と同じままにする。これを達成するために、カバレッジアレイループ制御装置１２３４は使用中のマスク１２３６を更新するとともに、新規なカバレッジ制御装置１２３８にカバレッジアレイのマスク１２３０を更新するように指示する。この第２の画素領域に対応するカバレッジアレイのアルファのエントリは、現在のフラグメントによっては変化しないので現在の画素領域（Aold）と同じままである。

【０４５１】

上記のように説明されたアプローチを繰り返すと、サブシステムは現在のフラグメントの使用中の各エントリを介してループして、各画素領域中の現在のフラグメントの効果がある場合は、それを計算する。そして、リストが空欄になるまで、リスト中の次のフラグメントの処理を繰り返す。

クランプ及び調整ブロック１２４８は累積されたカラーを適切な範囲（これは、カバレッジスケーラ部で丸められる必要があり、その結果８ビットの範囲を超えるカラーとアルファをもたらす）にクランプするとともに、１を示す８ビット２進数で値をスケーリングすることにより生じる誤差を調整する。この方法による誤差の調整は、値１が実際には１６進法値「ＦＦ」で表されることが必要である場合がある。換言すれば、０から１のアルファの範囲は００からＦＦの８ビット数の範囲で表される。それ故、数ｘにＦＦを乗算するときは、結果はｘである。この調整は、ＦＦの乗算結果が適切にｘに丸められる。

【０４５２】

画素バッファへのフィードバック経路１２５０は解析された画素値が画素バッファに再び格納されるモードをサポートするために存在し、その結果、解析したデータのチャンクをタイラーから共用メモリに送出することなく、解析した画素データ上でマルチパス・レンダリングを可能とする。

フラグメント解析サブシステムがフィードバックモードにないとき、クランプ及び調整ブロック１２４８は、図３６に示すように解析された画素データをデータ経路１２５２を経由してブロック・段バッファに送出する。これらのブロック・段バッファは、解析された画素データをそれが８×８画素ブロックで圧縮される前にバッファするのに使用される。

【０４５３】

テクスチャ・マッピング

この画像処理装置は、多くの進歩的なテクスチャ・マッピングの特徴を有している。テクスチャ・マッピングをサポートする特徴は、テクスチャデータの異方性フィルタリングを含む。この装置はリアルタイムでテクスチャデータの異方性フィルタリングを実行できる。

以下、異方性フィルタリングに対する本発明に係る解決法のための基礎を形成する幾つかの概念を説明することから始めて、その次に実施例を詳細に説明する。

テクスチャ・マッピングは、画像を面にマッピングすることに帰する。オブジェクトの面における複雑なディテールは、ポリゴン又は他の幾何形状プリミティブを用いてモデルを作ることは極めて難しく、且つそうすることはオブジェクトの計算コストを大幅に増大する。テクスチャ・マッピングは、画像処理装置がオブジェクトの面上のディテールを充分に表現することを可能にする。テクスチャ・マップは、ディジタル画像であり、それを「ソース画像」と規定する。このテクスチャ・マップは、形が典型的に矩形であり且つそれ自信の（ｕ，ｖ）座標空間を有する。テクスチャ・マップの個別の要素は、「テクセル」と呼ばれる。テクスチャ・マッピングにおいては、テクスチャ、即ち「ソース画像」を、目標画像に対してマッピングする。

【０４５４】

ディジタル画像としてのソース画像及び目標画像を、通常、整数座標を有する点の格子上の分離した点でサンプリングする。ソース画像においては、テクセルが前記の（ｕ，ｖ）座標系内の整数座標に置かれる。同様に、目標画像においては、画素が（ｘ，ｙ）座標系内の整数座標に置かれる。

幾何形状変換がいかにしてソース画像からの点を目標画像へマッピングするかを説明する。この逆変換が、いかにしてターゲット内の点をソース画像へマッピングし返すかを詳述する。画像処理装置は、テクセルのソースアレイ内のどこから画素濃度がこなくてはならないかを決めるためにこの逆変換式を用いることができる。ソース画像内のこの点での濃度を、その時付近のテクセルデータに基づいて決めることができる。ソース画像内へマッピングし返されたターゲット内の点は、テクセルの整数座標上に正確にある必要はない。この点における濃度を見つけるために、画像データが付近のテクセルから計算される。

【０４５５】

ソース画像濃度が、離散点の値で知られるのみであるから、付近のテクセルからの値は補間され且つ結果のデータがそれからローパスフィルタを通される。一般に、その解決法は次のように生じる。最初に、目標画像からソース画像内へ点がマッピングされる。それから、テクセルデータが、ソース画像内へマッピングされた点で、濃度を修正するように補間される。最後に、個々の目標画像内で正確にリサンプリングされるべき領域を、高すぎる領域へと変換しうるソース画像内の空間周波数を除去しておくために、ローパスフィルタが適用される。このローパスフィルタは、リサンプリングの故のターゲット内の低周波数の波として、見かけ上の高周波数、即ち「エイリアス」を除去するので、このローパスフィルタは、しばしばアンチエイリアシングフィルタと呼ばれる。以下に更に詳細にこの概念を説明する。

【０４５６】

図３７は、目標画像の面１３０２上の画素１３００が、テクスチャ・マップ１３０４の面に、いかにマッピングするかを示す一例である。この例においては、目標画像からの画素が正方形１３０６として表現されている。テクスチャ・マップ1304上へと向かう、この画素１３００の後方マッピングは、目的の面１３０２の曲率により、画素がマッピングできる形状に、より複雑な形状を近似する四角形１３０８である。テクスチャへ画素１３００をマッピングした後に濃度値がその四角形内のテクセルサンプルから計算される。例えば１つの解決法において、画素の濃度値がその四角形内のテクセルの加重和をとることにより計算される。

【０４５７】

補間及びローパスフィルタ機能の双方は、単一フィルタに組み合わせることができ、単一フィルタは、ターゲット内の離散点へマッピングする、ソース内の各逆変換点を取り囲む点について加重平均を取ることにより実行される。ここで、フィルタのフットプリントとして、その加重平均に寄与する点の領域を定めている。一般に、そのフットプリントは各ターゲット点に対するソース内で異なる形状を有する。そのフットプリントが各点に対して変わり得るので、そのフットプリントの正しい形状とそのフットプリントの内側の点

へ適用するための重み付け係数とを見つけることは難しい。幾つかの従来のシステムは、フィルタのサイズが変化することを許容するものであるが、全ての点に対して、フィルタとして同じ形状を使用する近似を行っている。しかしながら、この解決法は、最終画像内に歪みを生じさせる。

【０４５８】

ここで、等方性フィルタを、可変サイズの正方形又は円形のフットプリントを形成するフィルタと定めることとする。円形は、全ての方向に対して同じ長さを有するので、円形は真に等方性である。また、正方形は、水平方向と垂直方向とに等しいサイズを有するので、実質的に等方性であると考えることができる。

等方性のフィルタリングは、むしろ粗い近似を用いるので、等方性のフィルタリングは歪みを生じさせる。実際のフットプリントが大きく伸長したソース画像の領域では、正方形又は円形のような実質的に等方性形状のフィルタは、たとえサイズが調整できるとしても、フットプリントに対して乏しい置換フィルタである。等方性フィルタは、１つの形状のみを有するので、等方性フィルタは伸長したフットプリント内のテクセルを、正確に捉えることができない。例えば、正方形フィルタは、一方向に伸長した四角形フットプリントから、テクセル値を正確にサンプリングできない。実際のフットプリントの外側のテクセルをサンプリングすると、ブラーリングを生じさせる。これに反して、フットプリント内のテクセルをサンプリングしないと、最終画像にエイリアシング故のちらつきを生じさせる。

【０４５９】

ＭＩＰ（multum in parvo;小型で内容豊富）マッピングと呼ばれる解決方法においては、多くのテクスチャ・マップが種々の分解能で記憶される。例えば、１つのテクスチャが５１２×５１２ テクセルに有る場合に、又は、２５６×２５６， １２８×１２８， ６４ ×６４， 等で、本システムはテクスチャを記憶している。ある画像処理システムは、そのテクスチャ内へマッピングされた画素のフットプリント上で、等方性フィルタに対して最良の適合を見つけるために、様々な分解能でこれらテクスチャ・マップを使用できる。その画像処理装置は、最初に、フットプリントがフィルタのサイズに対して最も近いサイズとなる条件で、二つのテクスチャを見つける。次に、その画像処理装置は、フットプリントを最も近くに適合させる二つのテクスチャに対して補間を実行し、２つの中間値を算出する。最後に、その画像処理装置は、その二つの中間値の間を補間して、この画素に対する値を見出す。

【０４６０】

ＭＩＰマッピングは、等方性フィルタに対する改善された結果を与えることができるが、それだけでは、ＭＩＰマッピングは、特にフットプリントがある方向に伸長する場合に、歪みを生じさせる。各点において、実際のフットプリントに対してより正確なフィルタとしては、幾何形状変換の逆変換によりフットプリント形状が歪まされた際に、実質的に等方性のリサンプリングフィルタによって畳込まれた、実質的に等方性の復元フィルタのカスケード接続によって構成させるものがある。この場合の歪みは、フィルタの高程度の異方性を生じさせる。幾何形状変換が、ある方向よりも別の方向に、画像をより縮小させる場合に、その逆変換は、目標画像内で最大に縮小する方向に沿って、ソース画像内のフットプリントを拡大、即ち伸長することになる。エッジに近い遠近像から平らな面を見てみると、この歪みが生じている。等方性フィルタリングでは、等方性フィルタが、伸長したフットプリント内のテクセル値を正しくサンプリングできないので、最終画像は、この例においては歪まされて現れる。

【０４６１】

異方性フィルタリング方法の一実施例は、次の二つのステップを含んでいる。即ち、1)フィルタ・フットプリントの最大伸長の近似方向を見つけるステップ、及び、2)実際のフットプリントをより正確に適合させる合成フィルタを生成するように、復元フィルタの出力に対してその近似方向に沿ってリサンプリングフィルタを適用するステップ、である。

最大伸長の方向は、目標画像からテクスチャ・マップまでのフィルタの後方マッピング

から導出することができる。例えば遠近法マッピングにおいて（そこではオブジェクトが消失点に向かって次第に消える）目標画像からテクスチャまでの遠近法マッピングのｎ×ｎ画素フットプリントは、四角形である。異方性の線は、最大伸長の方向を有し、且つ、ソース画像内へマッピングし返されたターゲットからの点を通過する線として規定される。

この実施例においては、本画像処理装置は、最大伸長の方向を見つけるために、テクスチャに対してフィルタ・フットプリントを後方マッピングする。次に、本画像処理装置は、最大伸長の方向に沿って補間フィルタ（上記に略述した復元フィルタ）を掃引する。目標画像に対する画素値を計算するために、本画像処理装置は、補間フィルタの出力に対しリサンプリングフィルタを適用する。

【０４６２】

一実施例においては、リサンプリングフィルタは、異方性の線に沿って適合された一次元ディジタルフィルタである。それ故に、特定の一次元フィルタとして本発明の範囲を制限しない。

本実施においては、補間フィルタは、二次元等方性フィルタである。リサンプリングフィルタについてのように、補間フィルタの特定の種類として本発明の範囲を制限しない。二次元等方性フィルタは、１つの可能な実施例にすぎない。この補間フィルタは、付近のテクセルデータからの値を用いて補間することにより、異方性の線に沿った位置における値を生成する。補間フィルタがソース画像に適用された画像での個々のテクセルデータの位置は、各位置において異方性の線における値を垂直又は水平方向の増分でステップさせ、補間することにより決定することができる。例えば、異方性の線が水平よりもより垂直である場合には、１つの解決法は、テクスチャの（ｕ，ｖ）座標系における垂直方向、即ちＶ方向にステップさせることである。同様に、異方性の線が垂直よりもより水平である場合には、別の解決法は、テクスチャの（ｕ，ｖ）座標系における水平方向、即ちＵ方向にステップさせることである。

【０４６３】

異方性の線に沿ってステップさせるための１つの可能な方法は、略最小伸長の長さでの均一な間隔で、この線に沿った個々の位置に補間フィルタを適用することである。特に、異方性の線に沿ったサンプリング位置が、画素中心がテクスチャ・マップ内へマップする点に置かれた中心サンプリング位置を有する最小延長の長さと略等しい距離で、均一な間隔とすることができる。一旦これらのサンプル位置が計算されると、等方性フィルタは、各位置に繰り返し適用することができる。例えば、等方性フィルタを、最小伸長の長さに依存するフィルタのサイズで、各サンプルに対して近いテクスチャサンプルで補間を実行するように、該サンプル位置に適用することができる。この方法を実行するための１つの特定の方法は、異方性の線に沿った個々の点の各々でトライ・リニア補間を実行することである。

【０４６４】

ディジタルフィルタへ補間フィルタの出力を適用した後で、結果として生じる画素値は、異方性の線に沿った補間フィルタの出力の加重平均値である。ここに、フィルタの特定の種類を説明しているが、復元及びリサンプリング機能を近似するために用いられるフィルタの種類を変えることができる。

図３８Ａ～３８Ｄは異方性フィルタリングの処理の一例を示している。図３８Ａ～３８Ｄはテクスチャ・マップ内のテクセル（１４００ａ～ｄ）を示しており、いかにして異方性フィルタを生成することができるかを示している。第１ステップは、目標画像内の画素位置に対するフィルタ・フットプリント上で、逆変換式を実行することによりソース画像内のフィルタ・フットプリントを計算することである。この例においては、テクスチャ1400ａ内のフィルタ・フットプリントを、四角形１４０２として図示している。

【０４６５】

次のステップは、ソース画像内へマッピングされた点で逆変換行列を近似することである。図３８Ｂに示されたこの例においては、この近似が平行四辺形１４０４により表現さ

れている。この平行四辺形が四角形形状フットプリントを近似する。一般に、この近似が逆幾何形状変換のためのヤコビアン行列を計算することより見出すことができる。図示の目的のためにこの例を単純化したしたが、その逆変換は、より複雑な場合にも同じ概念を拡大解釈できる。これが以下に与えられた付加的な詳細な説明から明らかになる。

再び図３８における例を参照して、復元及びリサンプリングフィルタのサイズは、ヤコビアン行列から導出できる。図３８Ｂにおいては、平行四辺形１４０４としてヤコビアン行列を表している。この平行四辺形の長さを、リサンプリングフィルタのサイズを決めるために用いることができる。この例においては、長さが最大伸長の方向１４０６に沿って測定され、その方向を異方性の方向とも呼ぶこととする。同様に、四角形の高さを、復元フィルタのサイズを決めるために用いることができる。この高さは、最小伸長の方向１４０８である。

【０４６６】

図３８Ｃは平行四辺形を近似する矩形１４０６を示している。この矩形のサイズは平行四辺形の高さと長さとに対応している。この矩形が実際のフィルタ・フットプリントの異方性を近似するために用いられる「棒状」フィルタを表現している。

図３８Ｄは、いかにしてこの「棒状」フィルタを計算できるかを図示している。復元フィルタのフットプリントは正方形１４０８により表現されている。この例においては、復元フィルタは正方形フットプリントを有し且つ従って本質的に等方性フィルタである。図３８Ｄ中の線１４１０により表現される異方性の線に沿った値を計算するために、値が異方性の線１４１０の線を取り囲むテクセル（１４００ｄ）から補間される。それ故に、その復元フィルタは、前述した補間フィルタである。このフィルタの出力はその時、リサンプリングフィルタを表現する一次元フィルタへ加えられる。異方性の線１４１０は、リサンプリングフィルタの方向を表現している。復元フィルタとして計算される値は、異方性の線に沿って通され、目標画像のための画素値を計算するために合計される。

【０４６７】

上記の解決法は種々の方法で実行できる。上記の解決法はハードウェア又はソフトウエアで実行できる。リアルタイム異方性フィルタリングをサポートするために、この方法はハードウェアで好適に実行できる。この解決法の一実施例はタイラー(Tiler) チップ上で実行される。

図９Ａ～９Ｃに図示されたタイラーにおいては、異方性フィルタリングが走査変換ブロックとテクスチャフィルタ・エンジンとでサポートされる。走査変換ブロックがソース画像内の点での逆幾何形状変換の偏導関数のヤコビアン行列を取ることにより、リサンプリング及び復元フィルタのための制御パラメータを計算する。そのヤコビアン行列は、その逆変換に対して最良となる局部的なアフィン近似の線形部分を表している。より詳細に言えば、それは所望のソース点の周りに集中された、その逆変換の二次元でのテイラー数列の一次部分である。

【０４６８】

テクスチャ座標から画面座標へのアフィン変換の線形部分は、２×２ヤコビアン行列Ｊを有し、画面座標からテクスチャ座標への逆変換は、ヤコビアン行列$Ｊ^{-1}$を有する。行列$Ｊ^{-1}$の２つの行ベクトルの長さは単位サイズの画素に対する平行四辺形の２辺の長さである。逆ヤコビアン行列における２つの行ベクトルの成分が平行四辺形の２辺の長さを決定する。

この変換は走査変換ブロックが各プリミティブを走査するように走査変換エンジンが評価する属性エッジの式の形を取る。以下の式が典型的である。

$$F_{S/W} = A_{S/W}x + B_{S/W}y$$

ここで、画素位置（ｘ，ｙ）において；

1)　$F_{S/W}$ は一様座標（ｗ）により分割されたテクスチャ座標（ｓ）の値である。

2)　$A_{S/W}$ はｘ座標に対する一様座標（ｗ）により分割されたテクスチャ座標（ｓ）の勾配の値である。

3)　$B_{S/W}$ はｙ座標に対する一様座標（ｗ）により分割されたテクスチャ座標（ｓ）

の勾配の値である。Ｆ，Ａ及びＢはプリミティブの走査開始点に対して全部正規化されている。走査変換ブロックはl/w，s/w及びt/w に対するエッジ方程式を評価する。

【０４６９】

逆ヤコビアン行列要素が平行四辺形の辺の長さと面積とを与える。近似する矩形の面積と矩形の長辺とが同じであり、矩形の短辺は（ｓ，ｔ）座標系におけるｘ及びｙ軸の間の角度のsin値 で乗算された平行四辺形の短辺である。

逆ヤコビアン行列に対する導関数は、各テクスチャ座標（ｓ，ｔ）におけるエッジ方程式のFs，As及びBsから直接導出する。

【０４７０】

【数１２】

$$w = \frac{1}{1/w} = \frac{1}{F_{1/w}}$$

$$s = w(s/w) = wF_{s/w} \qquad t = w(t/w) = wF_{t/w}$$

$$\frac{\partial s}{\partial x} = w\left(\frac{\partial(s/w)}{\partial x} - s\left(\frac{\partial(1/w)}{\partial x}\right)\right) = w\left(A_{s/w} - sA_{1/w}\right) \qquad \frac{\partial s}{\partial y} = w\left(\frac{\partial(s/w)}{\partial y} - s\left(\frac{\partial(1/w)}{\partial y}\right)\right) = w\left(B_{s/w} - sB_{1/w}\right)$$

$$\frac{\partial t}{\partial x} = w\left(\frac{\partial(t/w)}{\partial x} - t\left(\frac{\partial(1/w)}{\partial x}\right)\right) = w\left(A_{t/w} - tA_{1/w}\right) \qquad \frac{\partial t}{\partial y} = w\left(\frac{\partial(t/w)}{\partial y} - t\left(\frac{\partial(1/w)}{\partial y}\right)\right) = w\left(B_{t/w} - tB_{1/w}\right)$$

$$\text{長さXの自乗} = \left(\frac{\partial s}{\partial x}\right)^2 + \left(\frac{\partial t}{\partial x}\right)^2$$

$$\text{長さYの自乗} = \left(\frac{\partial s}{\partial y}\right)^2 + \left(\frac{\partial t}{\partial y}\right)^2$$

$$\text{面積} = Abs\left(\left(\frac{\partial s}{\partial x}\right)\left(\frac{\partial t}{\partial y}\right) - \left(\frac{\partial s}{\partial y}\right)\left(\frac{\partial t}{\partial x}\right)\right) = L \times S \sin\theta$$

【０４７１】

ヤコビアン行列を見出した後に、その走査変換ブロックが２つの行ベクトルの長い方を見出す。このベクトルの方向が最大伸長の線の方向、即ち異方性の線の方向を表す。この行ベクトルの長さの他方のベクトルの長さに対する比率を、異方性比率と称する。一次元異方性フィルタの長さはこの比率から決められる。異方性比率により除算された長い方のベクトルの長さが復元フィルタの幅を制御する。

【０４７２】

長い方の辺は、主軸となり、且つテクスチャ座標においてステップさせる（クロックする）際に、その長い方の辺を、増大されるべき画面座標を決めるために用いることができる。その長い方の辺を、増分の大きさの決定のためにも用いることができる。

```
// DsDxはｘ等に対するｓの偏導関数である。
// (DsDc, DtDc)は異方性の軸に沿った（ｓ，ｔ）座標でのステップである。
if (LengthXSquared >= LengthYSquared) {
```

```
   MajorSquared = LengthXSquared
   InverseMajor = 1./sqrt(MajorSquared)
   DsDc = DsDx * InverseMajor
   DtDc = DtDx * InverseMajor
} else {
   MajorSquared = LengthYSquared
   InverseMajor = 1./sqrt(MajorSquared)
   DsDc = DsDy = InverseMajor
   DtDc = DtDy = InverseMajor
}
```

【０４７３】

ステップサイズDsDc及びDtDcは、テクスチャフィルタ・エンジンへの基本的入力であり、そのテクスチャフィルタ・エンジンがサンプリングとフィルタリングとを実行する。これらのステップは、（多くとも）７度分だけ正しくない方向を生成し、それは等辺平行四辺形の場合に起こる。

この実施においては、短い方のベクトルの長さは、異方性比率がプリセットされた制限を越えない限り、復元フィルタの幅を、通常、近似する。この制限を越えた場合には、異方性比率はその計算中にこのプリセットされた制限により置き換えられる。この方法において異方性比率を制限することが、フィルタが値を計算するためのテクセル点の予め決められた数よりも多く用いるのを防止する。従って、その異方性比率を制限することは、出力値を計算することを必要とする復元フィルタが、どれほどの長さであるかについての境界を置くことになる。

【０４７４】

いずれかのベクトルの長さが、１よりも小さい場合に、それとは別に制限する場合が生じる。その場合においては、ベクトルの実際の長さが１の長さにより置き換えられる。このことは、フィルタ長さが補間を実行するために決して短くなりすぎないことを確実にする。

走査変換ブロックがそのフィルタに対する制御パラメータを計算した後に、走査変換ブロックはそれから画素値を計算する。一次元ディジタルフィルタが補間フィルタからの出力の加重平均を計算する。その補間フィルタが、異方性の線に隣接するソース画像から、テクセルデータを補間することによりこの出力を計算する。

【０４７５】

この補間フィルタのサイズを、最大伸長の方向と垂直な方向で測定されたフットプリント幅を近似するように調整することができる。フットプリントが大きい場合（変換処理が縮小である際の画像の領域内に生じる）には、ソース画像の多くの点が単一出力点を生成するためにフィルタ重み付け係数で乗算されねばならず、変換処理が極めて遅い、即ちコストの高い実現例となる。

【０４７６】

上述したように、現存する等方性フィルタリング装置は、ＭＩＰマッピングを用いることにより計算時間を低減させる。ＭＩＰマッピングは、ソース画像に基づいて画像ピラミッドを形成することに帰し、且つその時ソース画像上で等方性フィルタに対する最良の適合を見出すために、このピラミッドの画像を用いる。このピラミッドの各レベルがサンプリング濃度でそれの下のレベルと比較して各サイズで２の係数分、低減される。このピラミッドの底は、元のソース画像である。低減されたサンプリング濃度の画像を補間することは、等方性フィルタ（フットプリントが、低減された濃度に対する元のサンプリング濃度の比によって、補間器のフットプリントに対して拡大される）で、元の画像をフィルタリングすることと同様の効果が生じる。従って、フットプリントの２のべき乗の拡大を、補間するためのピラミッドの正しいレベルを選択することにより達成することができる。拡大の如何なる比率も、所望の比率を跨る、２つのピラミッドレベルについての補間の結果を混合させることにより得ることができる。

【０４７７】
一実施例においては、等方性フィルタのサイズを、ＭＩＰマッピング解決法を用いることにより、最小伸長の長さに対して、より細かく適合するように修正することができる。ヤコビアン行列を解析することにより決められた等方性フィルタサイズは、混合させる二つのピラミッドレベルと混合係数とを選択するように用いることができる。一実施例において、基本ピラミッドレベルは、フィルタサイズにおける２を底とする対数の整数部であり、且つ混合係数は小数部である。
特定の例が上述の特定の実施の動作を図示するのを助ける。所望の等方性サイズが３である場合は、$\log_2 3$は１．５８３と等しい。この結果の整数部は１であり、その整数部が、それぞれ２及び４の濃度低減となるレベル１及び２を選択する。レベル０は低減無しとなる元のソース画像である。混合係数は０．５８５である。
一実施例において、テクスチャフィルタ・エンジンは、混合することを延期する。最初に、テクスチャフィルタ・エンジンが、各レベルにおいて所望の点に集中される異方性比率と比例する長さの１次元フィルタを適用する。それからテクスチャフィルタ・エンジンが各レベルからの出力を混合する。
別の実施例においては、テクスチャフィルタ・エンジンは、異方性の線に沿ってステップし、且つこの線に沿って分散するサンプルにおいてトライ・リニア補間を実行する。テクスチャフィルタ・エンジンが、それから各サンプルにおいてトライ・リニア補間の結果に対して一次元フィルタを適用する。
【０４７８】
補間フィルタのサイズを制御することに加えて、リサンプリングフィルタのサイズもまた制御することができる。一実施例においては、テクスチャフィルタ・エンジンは、種々のサイズの１次元リサンプリングフィルタのための係数の表を用い、且つ表内の特定されたサイズの間のサイズのフィルタを作るために、それらの間で混合する。高速ハードウェアに対して特に有効な一実施例は、２の累乗としてフィルタ長さを選択することであり、且つ三角あるいは台形形状を有するようなフィルタ・インパルス・プロファイルを選択することである。個別のフィルタは、極めて単純な係数を有し、その乗算の効果は、ハードウェアにおける少しの加算部とシフト部の使用へと低減させることである。
【０４７９】
以下は、最初の２の４乗までの、個別のフィルタ用の係数の表である。
【０４８０】
【表１９】

0								1								/1
1							1	2	1							/4
2					1	2	3	4	3	2	1					/16
3	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1	/64
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	/15

【０４８１】
この例においては、異方性比率の２を底とするlog がレベル及び混合係数を選ぶために用いられている。レベルが４を越えると、テクスチャフィルタ・エンジンが最後のフィルタを用い、且つ混合しない。この例においては、全部のフィルタが１の利得を有し、全部のそれらの係数が１へ加えることを意味している。１，２，４及び８による乗算はシフト動作により実行される。３，５及び６による乗算は、一回の加算に１回のシフト動作を加えることより実行することができる。最後に、７による乗算は一回の減算と複数のシフト動作とにより実行することができる。２の累乗による除算はシフトするだけである。１５による除算は、２を底とする１．０００１０００１による乗算と次の４つ分のシフト（１

６による除算）により極めてよく近似することができる。乗算は二つの加算のみである。
【０４８２】
上述の解決法は、合成フィルタの制御に二つの自由度を可能にする。上述の実施例においては、自由度は、最小及び最大伸長の方向におけるフィルタの大きさである。この解決法が、極めて非線形となるマッピングの結果とできる各点での真のフットプリントを計算する処理負担無しに、極めて少ないエイリアシング及びブラーリングを有する画像を生成する。この解決法は、異方性の線に沿って実際のフットプリントフィルタを掃引する連続フィルタを近似する。その解決法は、異方性の線に沿った「棒状」フィルタを生じさせるので、その解決法は、円形又は正方形よりも実際のフットプリントの極めて良好な適合性を達成する。そこで、リアルタイム・グラフィックス・レンダリング・システムに、この方法を実施した。それ故に、この方法がまだリアルタイム速度を維持しながら、異方性フィルタリングによる高品質テクスチャ・マッピングをサポートし、即ち10Hzより大きい速度で、且つ特に表示装置のリフレッシュ速度（例えば７５Ｈｚ）で画像データの新しいフレームを計算する。
【０４８３】
図３９は、テクスチャフィルタ・エンジン（４０１，図９Ｂ）の一実現例を図示するブロック図である。このテクスチャフィルタ・エンジンは、画素キュー（図９Ｂにおけるテクスチャ参照データ・キュー３９９；Pixel Queue）からテクスチャ参照データ例を読み取り、且つアルファ及びカラー値（アルファ、及びＲＧＢカラー係数）又はこれらテクスチャ参照データ例に対するシャドウ係数を計算する。この実現例では、テクスチャフィルタリングとシャドウフィルタリングとの双方をサポートする。テクスチャ・マッピング動作に対して、テクスチャフィルタ・エンジンは、テクスチャカラーとアルファとを計算し、テクスチャカラーをフィルタリングして、アルファ係数とカラー係数とを計算する。シャドウイング動作に対して、テクスチャフィルタ・エンジンが深度比較を実行し、その比較結果の値をフィルタリングして、シャドウ減衰係数（ｓ）を計算する。
【０４８４】
画素キューが（図９Ｂ内の走査変換ブロック３９５ ような）ラスタライザからテクスチャ参照データを受け取り、且つテクスチャフィルタ・エンジン４０１ に対するＦＩＦＯバッファとして働く。この「サンプル有効」データが、テクスチャキャッシュからフェッチされた１組のテクスチャ又はシャドウマップ要素において、どのサンプルが現在のフィルタリング動作に対して有効であるかを特定する。
【０４８５】
テクスチャ・マッピング動作に対して、テクスチャ参照データはテクスチャ（ｓ，ｔ）内へマッピングされた画素位置の座標を含んでいる。トライ・リニアＭＩＰマッピングをサポートするために、テクスチャ・マッピングの入力は、二つの最も近いMIPマップレベル(ｈｉ，ｌｏ)とディテールのレベル（ＬＯＤ）に対する（ｓ，ｔ）座標を含む。「累積スケール」データは、カラー成分補間器の出力に加えられる重み付け係数を制御するように用いられる。「拡張制御」データはテクスチャ拡張モードを制御するデータビットである。テクスチャ拡張モードはテクスチャ要求がテクスチャ・マップ範囲の外側にある場合に、クランプ、ラップ又は反射の動作のいずれかを実行するためにテクスチャフィルタ・エンジンに指令する。
シャドウイング動作に対して、入力はサンプルインデックス、シャドウマップ内へマッピングされた画素位置の（ｓ，ｔ）座標、及び所定の画素位置に対する光源からの幾何形状プリミティブの深度を表すベータを含んでいる。このサンプルインデックスは、シャドウフィルタがシャドウマップ要素、即ち「サンプル」上で動作する特定の方法に関係している。この特定の実施では、テクスチャフィルタ・エンジンがクロック・サイクル当たり８サンプルに動作する。シャドウフィルタリングの場合には、これらのサンプルが４×２グリッドに対応する。例えば、シャドウフィルタは４×４モード（４×２＋４×２＝４×４）に対する２組のサンプルと８×８モードに対する８組のサンプルとの全部に動作する。４×４モードの場合には、シャドウフィルタが４×４の全体フットプリント内の上側左

、上側右、下側左、及び下側右の３×３ブロックへ各々１個に、３×３フィルタを４回適用する。第１クロック・サイクルにおいては、それが上側の４×２グリッドを処理し、且つ第２クロックにおいてはそれは下側の４×２グリッドを４×４ブロックにおいて処理する。このサンプルインデックスは、現在処理されている８組のサンプルを識別するように用いられるインデックスである。このサンプルインデックスは、４×４の場合に対しては２クロック・サイクルによって、また８×８の場合に対して８クロック・サイクルによってステップし、且つどの４×２部分集合が現在処理されているかを識別する。

【０４８６】

図４１に示されるように、テクスチャフィルタ・エンジンは、キー生成器１３１０、端数制御器１３１２、カラー成分補間器１３１４、シャドウフィルタ用アキュムレータ１３１６、及び、累積及び後処理器１３１８を含んでいる。

テクスチャ・マッピング動作において、キー生成器１３１０は、（ｓ，ｔ）座標とＬＯＤとを読み取り、且つテクスチャキャッシュから対応するテクスチャデータをフェッチするためにキャッシュキーを生成する。テクスチャキャッシュは、テクスチャ要求に応じてアルファとＲＧＢ成分とを返す。端数制御器１３１２が入力として（ｓ，ｔ）座標を受け取り、且つカラー成分補間器１３１４においてバイ・リニア及びトライ・リニア補間器としての動作の双方又はいずれか一方を制御する。カラー成分補間器１３１４が補間さたアルファとＲＧＢ成分とを計算するためにテクセルサンプルを補間する。累積及び後処理器１３１８がそれからアルファとＲＧＢ成分とをスケーリングし、スケーリングされた成分を累積し、且つ現在処理されている画素位置に対応するアルファとカラー係数とを出力する。これらのアルファ係数及びカラー係数は、テクスチャ調整処理の入力値として、画素エンジンへのカラー値及びアルファ値の入力値となる。

【０４８７】

異方性テクスチャ・マッピングにおいて、カラー成分補間器１３１４が異方性の線に沿って進み、且つ各ステップにおいてトライ・リニア補間を実行する。アキュムレータ１３１８が一次元フィルタとして働き、アルファとカラー成分とをスケーリングし、且つそれからスケーリングされた成分を累積する。１つの特定の実施例においては、アキュムレータ１３１８が異方性の比率に基づいた台形又は三角形フィルタリングを用いてアルファ及びカラー成分をスケーリングする。双方の場合に、フィルタのエッジにおけるロールオフを近似するためにリサンプリングフィルタの遠端での成分をアキュムレータがスケーリングする。台形フィルタリングを達成するために、スケーリング係数はフィルタのエッジにおける線形のロールオフに対応し、且つフィルタのエッジの間のステップにおいて一定である。

【０４８８】

ある特定の実現例において、異方性の線に沿ってステップさせるためのスケーリング係数は、次のように計算される。１対１から２対１までの異方性比率に対して、このアキュムレータは、異方性のウオーカーの各ステップ動作において0.5 の重み付け係数を適用する。２対１及びそれより大きい異方性比率に対しては、アキュムレータは、ステップｎ＜（異方性－１）／２に対して１／異方性による成分を重み付けし、且つ（異方性－１）／２より大きいか又はそれに等しいｎに対して 0.5（異方性－２ｎ）／異方性による成分を重み付けする。この特定の例における異方性比率は、逆ヤコビアン行列に対して最もよく適合する矩形の短辺に対する長辺の比である。逆ヤコビアン行列は、ビュー空間座標からテクスチャ座標（すなち（ｘ，ｙ）座標から（ｓ，ｔ）座標）への幾何形状変換の偏導関数の行列である。異方性の線は、逆ヤコビアン行列の長い方の列ベクトルの方向における（ｓ，ｔ）座標を通る線である。

シャドウイング動作に対して、キー生成器１３１０がシャドウマップ内へマッピングされた画素位置の（ｓ，ｔ）座標を読み取り、且つキャッシュキーを生成する。テクスチャキャッシュが、シャドウフィルタアキュムレータ１３１６にシャドウマップ要素（シェイデル ； shadel）を返す。シャドウフィルタが入力としてシャドウインデックスとベータとを受け取り、且つシャドウマスクを生成するためにフィルタ・フットプリント内の深度

値と光空間における現在の画素データ例の深度とを比較する。シャドウフィルタアキュムレータがシャドウマスク内の要素を合計し、且つサンプルの数によりその合計を除算する。この実行において、テクスチャフィルタ・エンジンが深度比較ステップの結果に対して台形フィルタを適用することにより、フィルタ・フットプリントのエッジにおいて円滑なロールオフを達成する。台形フィルタを実行するために、シャドウ累積フィルタがそれぞれ４×４又は８×８フィルタ・フットプリントに対して４回３×３又は７×７ボックスフィルタを適用することにより、４個の予備シャドウ係数を計算し、且つカラー成分補間器１３１４のうちの１つへその４個の予備係数を送る。このカラー成分補間器１３１４は、最終のシャドウ係数を計算するためにそれら予備係数でバイ・リニア補間を実行する。

【０４８９】

上に紹介したように、キー生成器１３１０は、画素キューから（ｓ，ｔ）座標を読み取り、且つテクスチャキャッシュからテクスチャデータをフェッチするためにキャッシュキーを生成する。図４０はより詳細にこのキー生成器を図示するブロック図である。ハイ・アンド・ロー・ ＭＩＰマップ（二つの最も近いＭＩＰマップ）内の（ｓ，ｔ）座標に基づいて、キー生成器は、そのハイ・アンド・ロー・ＭＩＰマップ内のテクスチャサンプル位置を計算する（１３４０）。次に、このキー生成器は、これらサンプルからキャッシュキーを計算する（１３４２）。このキー生成器は、キャッシュキー、（ｓ，ｔ）座標、及びハイ・アンド・ロー・ ＭＩＰマップレベル用のＬＯＤを、要求されたテクスチャサンプルを返すテクスチャキャッシュに転送する。当然に、ディテールについての１つだけのテクスチャ・マップレベルが用いられる場合には、このキー生成器は、１つのテクスチャ・マップに対してのみキーを生成する。

図３９における端数制御器１３１２は、テクスチャ又はシャドウマップ内のサンプル間及びトライ・リニア補間のためのＭＩＰマップレベル間の補間を制御する。バイ・リニア補間をサポートするために、端数制御器がテクスチャ又はシャドウマップ内のサンプル間の重み付けを制御する。トライ・リニア補間をサポートするために、端数制御器が２つの最も近いＭＩＰマップレベル内にマッピングされた点に対して４個の最も近いサンプルの間で補間すること（バイ・リニア補間）を補間器に指令し、それから二つのＭＩＰマップレベルからの結果を混合するために線形補間を指令する。この端数制御器は入力としてハイ・アンド・ロー・ ＭＩＰマップレベルに対するＬＯＤ及び（ｓ，ｔ）座標を受け取り、且つ各MIPレベルにおけるサンプル間とＭＩＰマップレベル間の補間を制御する。

【０４９０】

カラー成分補間器１３１４は、アルファ及びＲＧＢカラー成分のための補間器を含んでいる。図４１は、４つの補間器のうちの１つを詳細に図示しているブロック図である。この補間器は１つの成分に対するカラー成分補間を取り扱い、且つシャドウ係数にバイ・リニア補間を実行する。他のカラー成分補間器は、カラー成分のみを取り扱う。

このカラー成分補間器は、テクスチャキャッシュからテクセル又はシャドウマップ要素を受け取り、且つそれらをマルチプレクサ（ＭＵＸ）のバンク１３５０へ加える。マルチプレクサのバンク１３５０へ入力する場合に、サンプル有効データは、どのサンプルが有効であるか、即ち現在のテクスチャ又はシャドウイング動作に用いられるべきサンプルを識別させる。サンプル有効制御信号に基づいて、それらマルチプレクサは、入力してくるサンプル又はテクスチャ背景カラー１３５２のいずれかを選択する。シャドウイング動作に対しては、カラー成分補間器１３１４は、シャドウフィルタアキュムレータ１３１６へシャドウ要素を送る。３つのカラーチャネルは、単一の24ビット幅シャドウマップ要素を形成するために用いられ、且つ、アルファチャネルは、シャドウイング動作において無視される。テクスチャ・マッピング動作に対して、カラー成分補間器は、テクスチャサンプルを線形補間器のステージ１３５４， １３５６及び１３５８に転送する。

【０４９１】

トライ・リニア補間においては、カラー成分補間器は、線形補間器の３つのステージを用い、２つのステージは、各ＭＩＰマップレベル（１３５４，１３５６）におけるサンプルの間の補間のためであり、且つもう１つのステージは、各ＭＩＰレベル（１３５８）か

らの結果を混合するためである。カラー成分補間器は、４つのフィルタ・フットプリントから計算されたシャドウ係数を組み合わせるために、バイ・リニア補間を実行する。図４３に示すように、このバイ・リニア補間を実行するために、最後の２つのステージ（１３５６及び１３５８）が用いられる。マルチプレクサの第２バンク１３６０は、４つのシャドウ係数と線形補間器の第１のステージ１３５４の出力との間での選択を行なう。テクスチャ・マッピングとシャドウ動作との双方において、カラー成分補間器は、補間器の最終ステージ（１３５８）の出力を、累積及び後処理器１３１８に転送する。

【０４９２】

シャドウフィルタアキュムレータ１３１６は、画素キューからサンプルインデックス及び光の深度の値（ベータ）を受け取り、シャドウマスクを生成するためにその光の深度の値をテクスチャキャッシュから返されたシャドウマップ要素と比較し、且つ予備シャドウ係数を計算するためにそのシャドウマスクをフィルタリングする。図４４は、より詳細に、シャドウフィルタアキュムレータを図示するブロック図である。シャドウフィルタアキュムレータ内の深度比較器がフィルタ・フットプリント内のシャドウ要素の深度を比較し、且つシャドウマスクを生成する。この特殊の場合において、シャドウマスクはフィルタ・フットプリントの４×２部分に対応する論理値による８ビットである。

【０４９３】

フットプリント制御器１３７２は、画素キューからサンプルインデックス値に基づいて全体フットプリントの現在の４×２部分を選択する。フットプリント制御器がクロック・サイクルとフィルタリングモード（２×２，４×４又は８×８）とに基づいて４個のシャドウ・コントリビューション部の各々に、フットプリントマスクを転送する。そのフットプリントマスクが８つのシャドウマスク要素のうちのいずれが、４×４及び８×８モードにおいて、４つのボックスフィルタの各々に対して現在のクロック・サイクルにおいて有効であるかを示す。２×２モードにおいては、シャドウフィルタアキュムレータ１３１６は、４つの最も近いサンプルの各々がシャドウ内に在るか否かを示す４つの論理値を出力する。

シャドウフィルタアキュムレータ１３１６は、フィルタ・フットプリント内のサンプルに対して４つのボックスフィルタ（例えば、３×３又は７×７）を適用する。シャドウマスクのどの要素が現在のクロック・サイクルに対して有効であるかを決め、次に、それら有効な要素を合計するために、シャドウ・コントリビューション部の各々がフットプリントマスクとシャドウマスクとを組み合わせる。全体のフィルタ・フットプリントに対するシャドウマスク内の有効な要素を累積した後に、シャドウ・コントリビューション部が予備シャドウ係数を計算するためにサンプルの数によりその合計を除算し、それらがカラー補間器内のバイ・リニア補間段へ転送される。次に、カラー補間器は、最終シャドウ係数を計算するために４つの予備シャドウ係数の間を補間する。

【０４９４】

累積及び後処理器１３１８は、カラー成分補間器１３１４からアルファ及びカラー成分を受け取り、且つテクスチャ参照データの各例に対してカラー係数及びアルファ係数を計算する。シャドウイング動作に対して、テクスチャフィルタ・エンジンはシャドウ減衰係数を計算するために１チャネル（アルファ又はＲＧＢ）を使用する。シャドウフィルタリング論理は別々にも実現できる。図４３は、より詳細に、累積及び後処理器を図示するブロック図である。同図のように、各カラー成分（アルファ及びＲＧＢ）は、スケール及び累積器１３８０を有している。各成分に対する、スケール及び累積器１３８０は、入力として累積スケールとカラー成分とを受け取り、且つ応答して、カラー成分をスケーリングし且つそれを成分合計ブロック１３８２内の累積された成分値へ加える。例えば、異方性フィルタリングにおいて、テクスチャフィルタ・エンジンが異方性の線に沿って進む時に、スケール及び累積器ブロック１３８０が復元フィルタ（トライ・リニア補間器）の出力を重み付けする。最後にステップした後に、アルファ及びＲＧＢ成分のためのスケール及び累積器が最終カラー成分係数を出力する。

【０４９５】

シャドウイング動作に対して、スケール及び累積器は、多数の動作をバイパスするが、その際に、周囲のオフセットを加える。この周囲のオフセットは、シャドウ内の全体的に平坦なオブジェクトがまだ見えることを確実にする。例えば、１のシャドウ係数が全体に照明されたことを意味し、０のシャドウ係数が全体にシャドウであることを意味している。カラー値が０の係数により乗算された場合には、オブジェクトは、その画素位置では見えなくなってしまう。そのため、オフセットが加えられ、且つシャドウ係数は、オフセットシャドウ係数がオフセット値から１までの範囲となるように、１にクランプされる。

シャドウ後処理器１３８４は、全部で３つのカラー値のチャネルに対して、及び（条件付きで）アルファ値のチャネルに対して、スカラーで表すシャドウの減衰値「ｓ」の模写を行なう。シャドウ画像を計算することのためにシャドウの減衰値を条件付き補数化（ｓ＝１－ｓ）とすることもある。シャドウ画像は、シャドウ係数のアレイであり、あるいはｇスプライトをシャドウイングするために用いることができる、シャドウ係数の補数のアレイである。

【０４９６】

最後に、マルチプレクサステージ１３８６は、シャドウイング動作のためのシャドウ係数か、又はテクスチャ・マッピング動作のためのＲＧＢ成分及びアルファ成分のいずれかを選択する。要するに、テクスチャフィルタ・エンジン４０１ は、シャドウイング動作とテクスチャリング動作との双方を実行する。テクスチャフィルタ・エンジンは、テクスチャ調整ステージへテクスチャ・マッピング動作の結果を送る。テクスチャ調整は、典型的には、補間されたカラー値、又は走査変換ブロックにおいて計算されたカラー値により、テクスチャフィルタからのＲＧＢカラー値を乗算することを含んでいる。さらに半透明性を有するグラフィカルオブジェクトに対して、テクスチャ調整は、走査変換ブロックからの補間されたアルファ値により、テクスチャフィルタからのアルファ値を乗算することを含むことができる。この実現例に依存して、テクスチャ調整は、テクスチャフィルタ・エンジン（図９Ｂにおける要素４０１）又は画素エンジン（図９Ｂにおける要素４０６）において実行することができる。それはまた、走査変換ブロック（図９Ａにおける要素３９４ 又は図９Ｃにおける要素３９７）においても実行することができる。一実施例においては、テクスチャフィルタ・エンジン４０１は、補間された値を、構成される値を計算するためにフィルタリングされた値と組み合わせる。画素エンジン４０６ は、その時構成された値を記憶するか又は対応するＲＧＢ成分又は対応する画素位置に対する画素値又はフラグメントバッファ内に記憶されたアルファ成分と組み合わせるか否かを決定する。

【０４９７】

シャドウイング動作の場合においては、画素バッファ内、又はフラグメントバッファ内の対応する画素位置におけるＲＧＢ及びアルファ値に対して、シャドウ係数を追加することができ、又は、現在の経路の間に生成され且つキュー内でバッファされた、補間されたＲＧＢ又はアルファ値に対して、シャドウ係数を追加することができる。例えば、オブジェクトが、ＲＧＢ及びアルファ値に関連するテクスチャを有さない場合には、テクスチャフィルタ・エンジン４０１ 内のテクスチャ調整ステージは、累積及び後処理器からのシャドウの減衰係数により、補間され且つ未解析のＲＧＢ及びアルファ値（照明された画像を表し、テクスチャ参照データ・キュー３９９(図９Ｂ)内に記憶されている）を乗算できる。

【０４９８】

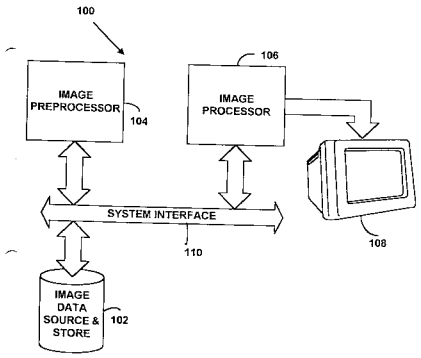
幾つかの実施例を参照して、画像処理システム、それの構造、及び関連する方法の種々の態様を説明した。詳細に幾つかの実施例を説明したが、これらの特定の実施例に本発明を制限するつもりはない。例えば、本発明の新規な構造は、手動の装置からワークステーションまでの規模のコンピュータシステム、ゲームプラットフォーム、セット・トップ・ボックス、グラフィックス処理ハードウェア、グラフィックス処理ソフトウェア、及びビデオ編集装置を含み、更に、それらに制限されない種々のハードウェア構成に適用することができる。本発明に係るシステム及び方法の変形例は、ハードウェア又はソフトウェア、或いはそれら双方の組み合わせで実現することができる。

【０４９９】
本発明の原理を置くことができる、多くの可能な態様の範囲において、上述された詳細な実施例は説明するためのみのものであって、本発明の範囲を制限するものとして取られるべきではないことを強調する。むしろ、特許請求の範囲に記載の請求項、及び、それらの請求項に対して等価な範囲と趣旨の範囲内に入るような全ての態様を、本発明として要求する。
【図面の簡単な説明】
【０５００】
【図１】画像処理システムのブロック図である。
【図２】本発明を実現するためのシステム環境を表わすブロック図である。
【図３】一実施例のシステム・アーキテクチャのブロック図である。
【図４Ａ】一実施例の画像処理ハードウェアのブロック図である。
【図４Ｂ】一実施例における幾何形状プリミティブをレンダリングする画像処理装置の一部を示すブロック図である。
【図５Ａ】一実施例におけるレンダリング処理の概要を示すフローチャートである。
【図５Ｂ】一実施例におけるレンダリング処理の概要を示すフローチャートである。
【図６】一実施例の表示生成処理の概要を示すフローチャートである。
【図７】一実施例における、フレーム周期に関して表示生成の一態様を示す図である。
【図８】一実施例におけるディジタル信号プロセッサ(ＤＳＰ) のブロック図である。
【図９Ａ】タイラーの別の実施例を示すブロック図である。
【図９Ｂ】タイラーの別の実施例を示すブロック図である。
【図９Ｃ】タイラーの別の実施例を示すブロック図である。
【図１０】は、メモリからテクスチャデータにアクセスするためのシステムを示すブロック図である。
【図１１】メモリからテクスチャデータにアクセスするためのシステムを示すブロック図である。
【図１２Ａ】ｇスプライト・エンジンの別の実施例のブロック図である。
【図１２Ｂ】ｇスプライト・エンジンの別の実施例のブロック図である。
【図１３】一実施例における合成用バッファのブロック図である。
【図１４】一実施例におけるディジタル・アナログ変換器(DAC) のブロック図である。
【図１５Ａ】チャンキングの一態様例を示す図である。
【図１５Ｂ】チャンキングの一態様例を示す図である。
【図１５Ｃ】チャンキングの一態様例を示す図である。
【図１６Ａ】一実施例におけるチャンキングの態様を示す図である。
【図１６Ｂ】一実施例におけるチャンキングの態様を示す図である。
【図１７Ａ】一実施例におけるチャンキングの態様を示すフローチャートである。
【図１７Ｂ】一実施例におけるチャンキングの態様を示すフローチャートである。
【図１８Ａ】一実施例におけるチャンキングの態様を示す図である。
【図１８Ｂ】一実施例におけるチャンキングの態様を示す図である。
【図１９Ａ】一実施例におけるチャンキングの態様を示す図である。
【図１９Ｂ】一実施例におけるチャンキングの態様を示す図である。
【図２０】一実施例における画像圧縮を示すブロック図である。
【図２１Ａ】一実施例におけるｇスプライト処理を示すフローチャートである。
【図２１Ｂ】一実施例におけるｇスプライト処理を示すフローチャートである。
【図２２】一実施例におけるｇスプライト変換を実行する方法の一態様を示す一実施例におけるｇスプライト処理を示すフローチャートである。
【図２３】一実施例においてｇスプライト変換がどのようにして伝達遅延を低減させることができるかを示す図である。
【図２４】一実施例におけるｇスプライト・データ構造のブロック図である。
【図２５】一実施例における、出力装置座標にマップされるｇスプライトの例を示す図で

ある。
【図２６】一実施例における表示生成の一態様を示す図である。
【図２７】26図の表示生成を、バンド周期という観点で示すフローチャートである。
【図２８Ａ】３通りの実施例における画素及びフラグメントの生成の態様を示すフローチャートである。
【図２８Ｂ】３通りの実施例における画素及びフラグメントの生成の態様を示すフローチャートである。
【図２８Ｃ】３通りの実施例における画素及びフラグメントの生成の態様を示すフローチャートである。
【図２８Ｄ】３通りの実施例における画素及びフラグメントの生成の態様を示すフローチャートである。
【図２８Ｅ】３通りの実施例における画素及びフラグメントの生成の態様を示すフローチャートである。
【図２８Ｆ】３通りの実施例における画素及びフラグメントの生成の態様を示すフローチャートである。
【図２９】本発明の一実施例における、画素フラグメントをマージする方法を示すフローチャートである。
【図３０】本発明の一実施例における、フラグメントマージ回路の一実現例を示すブロック図である。
【図３１】第30図に示すフラグメントマージ回路のマージテスト・モジュールの一実現例を示すブロック図である。
【図３２】画素及びフラグメントバッファの一部を示す図である。
【図３３】このピラミッド構造を表わす図である。
【図３４Ａ】タイラーにおけるバッファ分割法を示すフローチャートである。
【図３４Ｂ】タイラーにおけるバッファ分割法を示すフローチャートである。
【図３５】フラグメント解析サブシステムの一実現例を示すブロック図である。
【図３６】フラグメント解析サブシステムの別の実現例を示すブロック図である。
【図３７】テクスチャ・マッピングを示す図である。
【図３８Ａ】一実施例における異方性フィルタの方法を示す図である。
【図３８Ｂ】一実施例における異方性フィルタの方法を示す図である。
【図３８Ｃ】一実施例における異方性フィルタの方法を示す図である。
【図３８Ｄ】一実施例における異方性フィルタの方法を示す図である。
【図３９】テクスチャ及びシャドウフィルタの一実現例を示すブロック図である。
【図４０】図３９のキー生成器の一実現例を示すブロック図である。
【図４１】図３９のカラー補間器の一実現例を示すブロック図である。
【図４２】図３９のシャドウフィルタ用アキュムレータの一実現例を示すブロック図である。
【図４３】図３９の累積及び後処理器の一実現例を示すブロック図である。

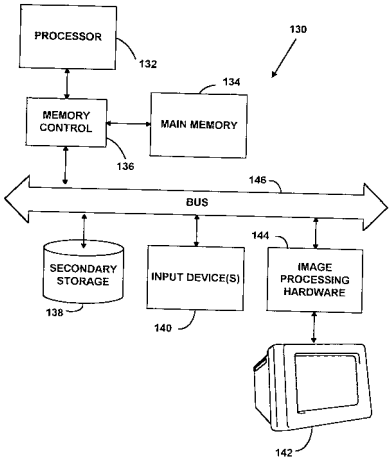
【図１】

FIG. 1



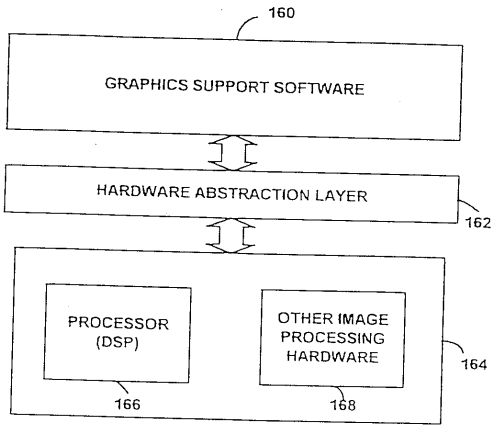
【図２】

FIG. 2



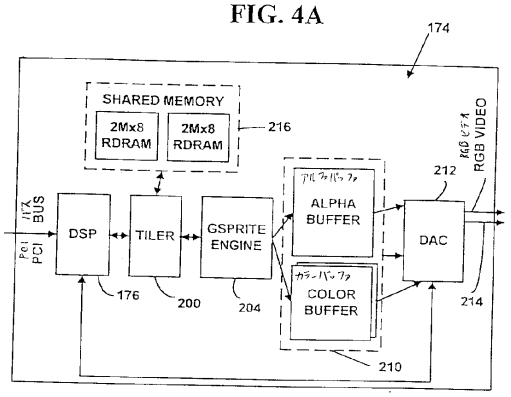
【図３】

FIG. 3



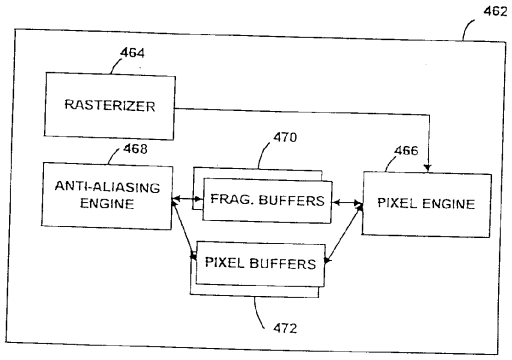
【図４Ａ】

FIG. 4A



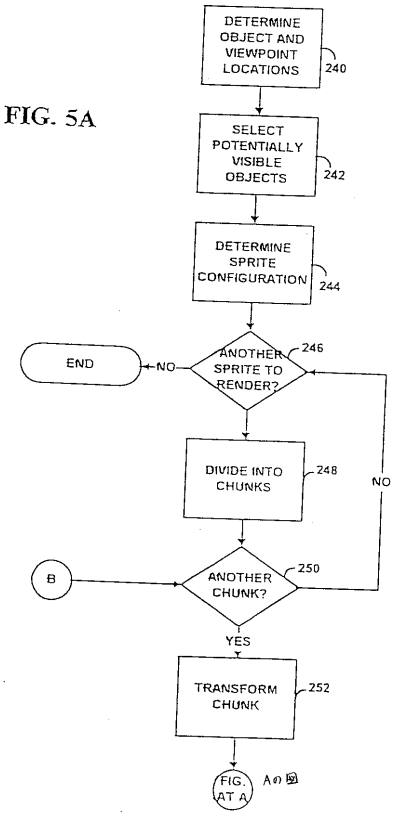
【図４Ｂ】

FIG. 4B



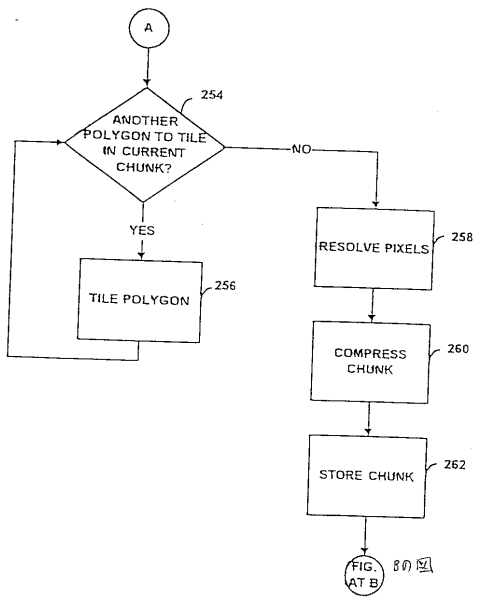
【図５Ａ】

FIG. 5A



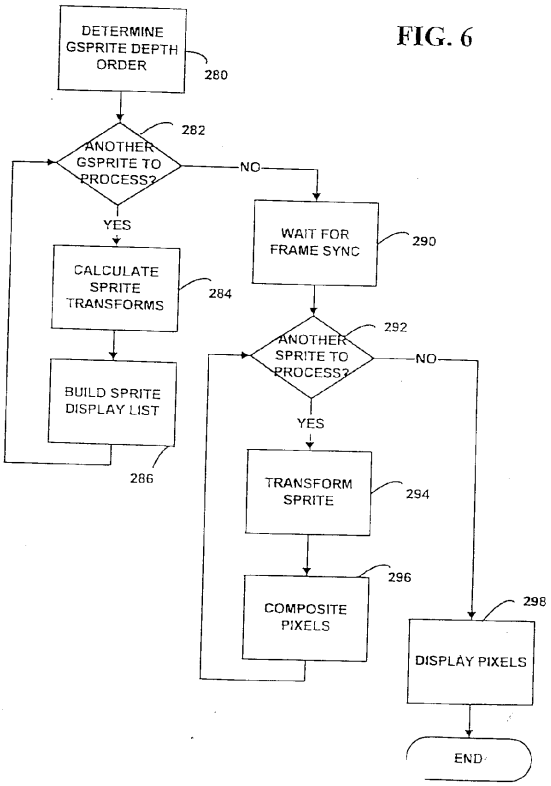
【図５Ｂ】

FIG. 5B



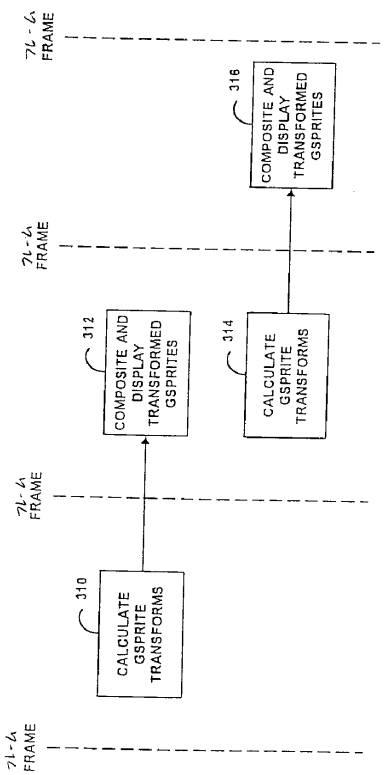
【図６】

FIG. 6



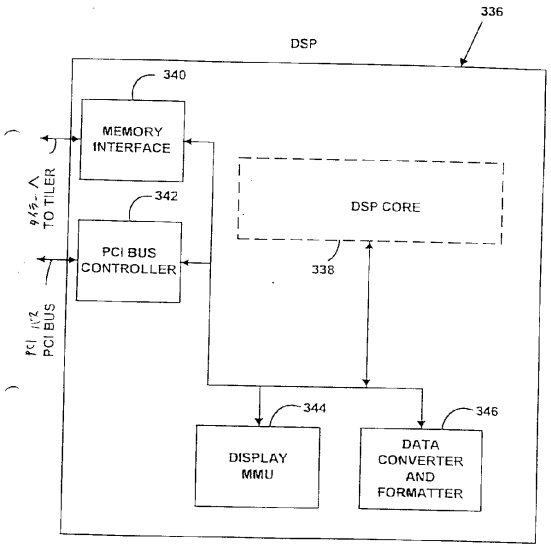
【図７】

FIG. 7



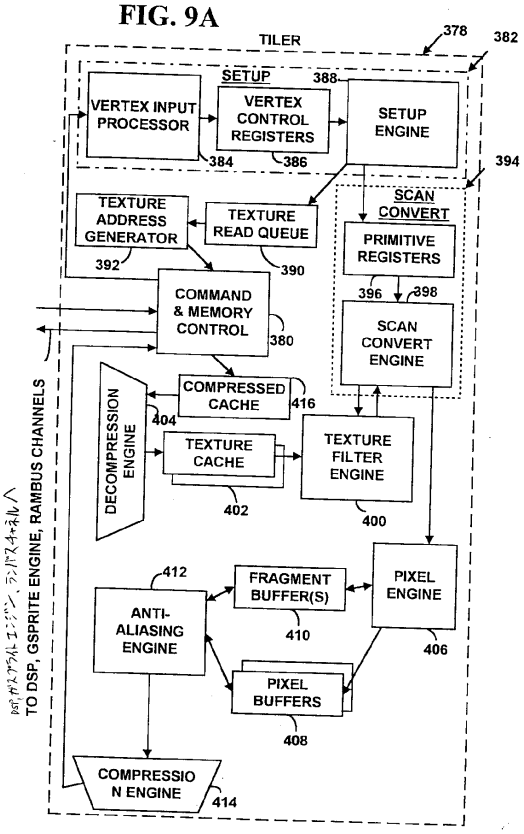
【図８】

FIG. 8



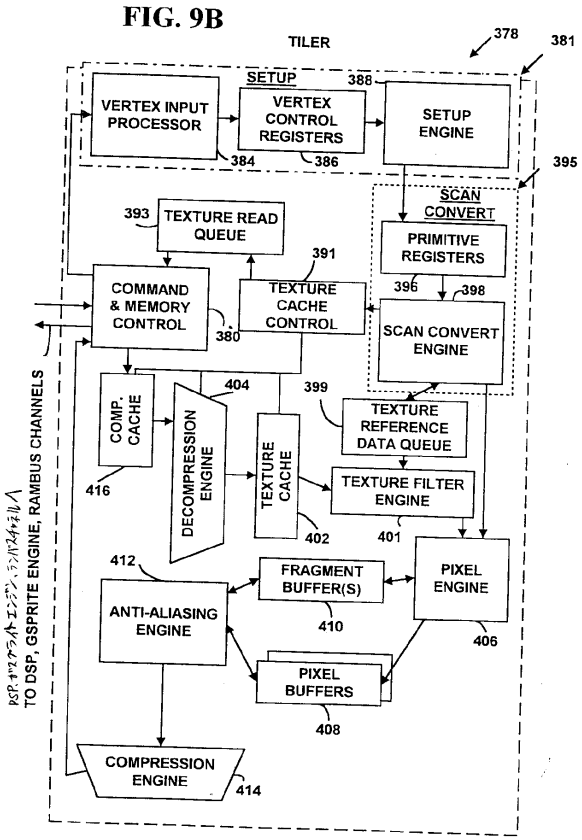
【図９Ａ】

FIG. 9A



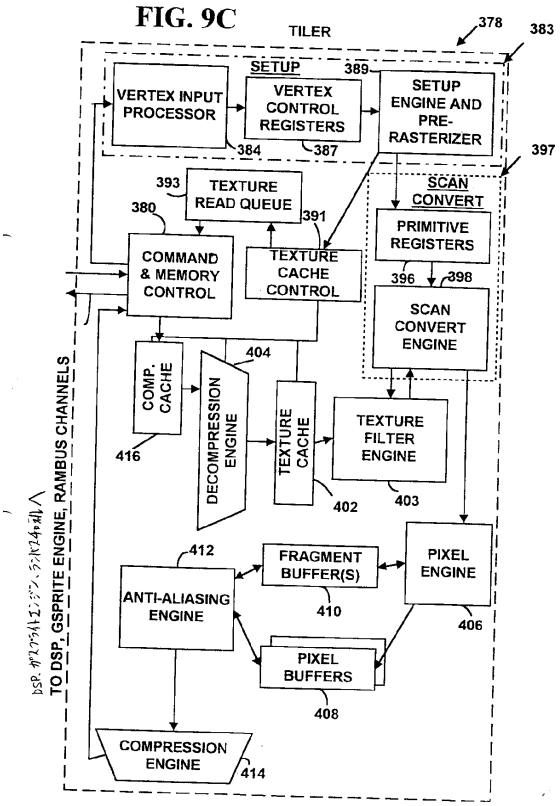
【図９Ｂ】

FIG. 9B



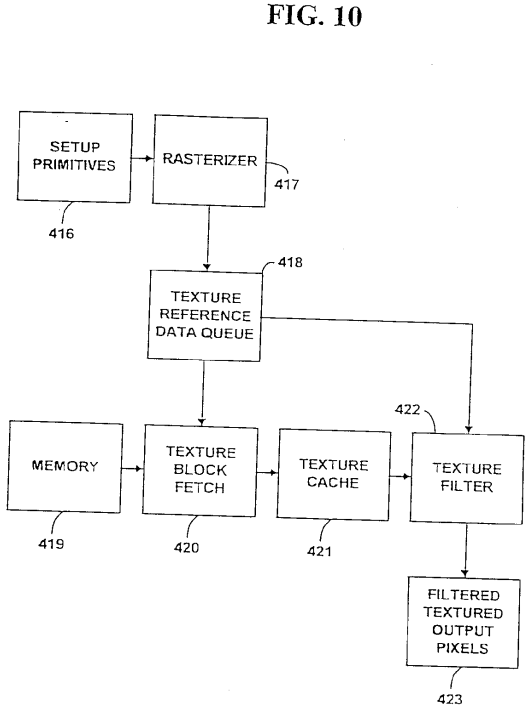
【図９Ｃ】

FIG. 9C



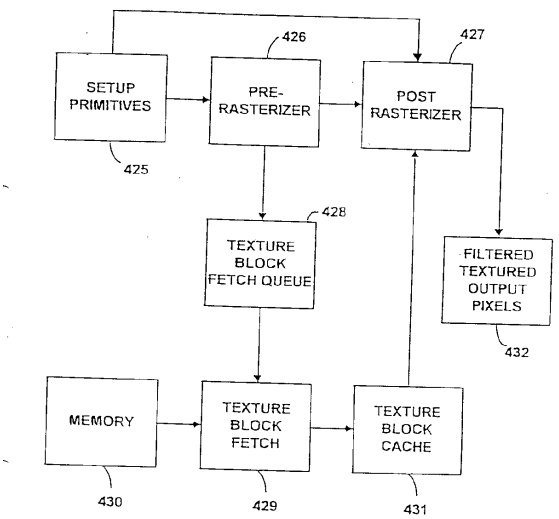
【図１０】

FIG. 10



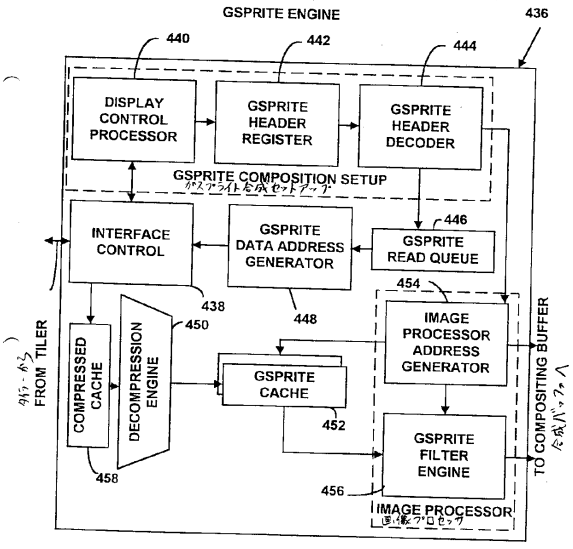
【図１１】

FIG. 11



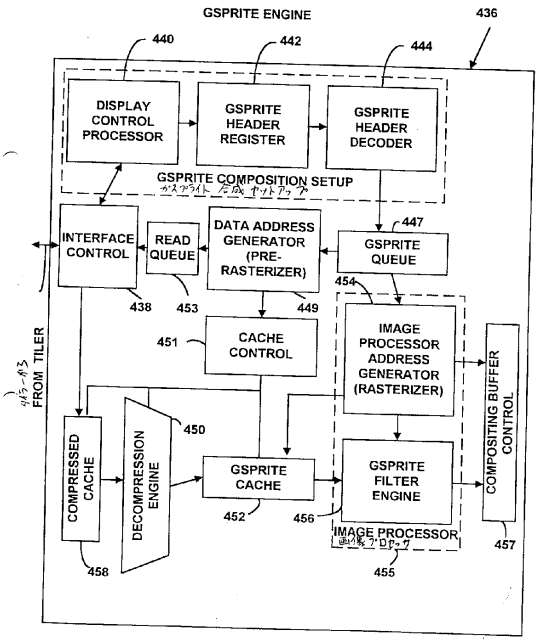
【図１２Ａ】

FIG. 12A



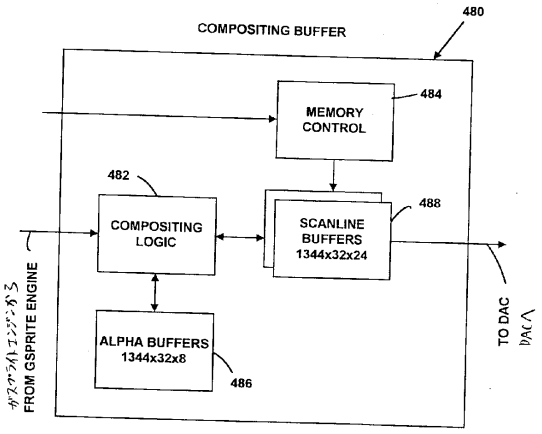
【図１２Ｂ】

FIG. 12B



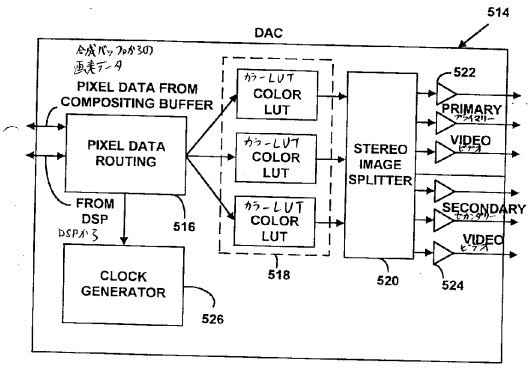
【図１３】

FIG. 13



【図１４】

FIG. 14



【図１５】

FIG. 15A

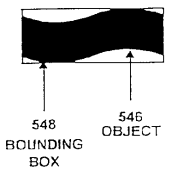


FIG. 15B

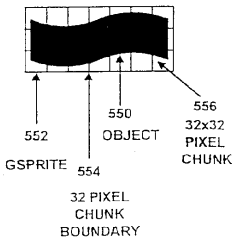
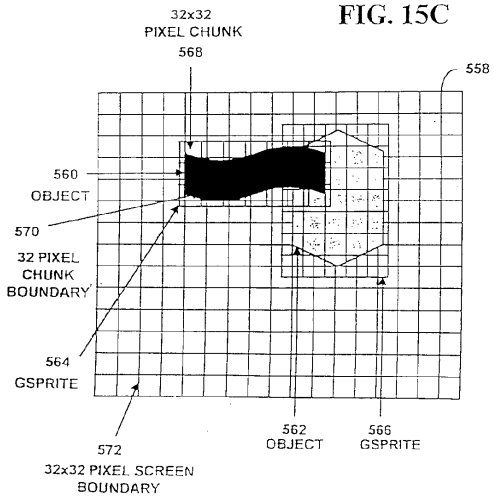


FIG. 15C



【図１６】

FIG. 16A

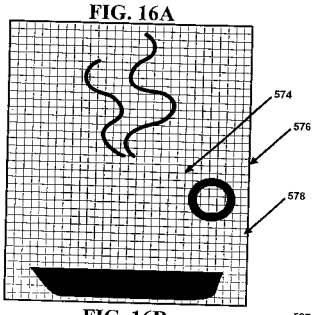
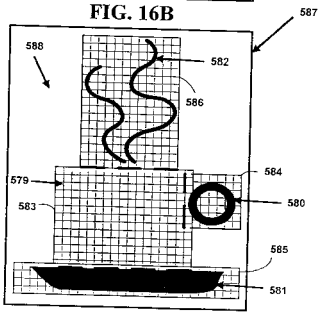
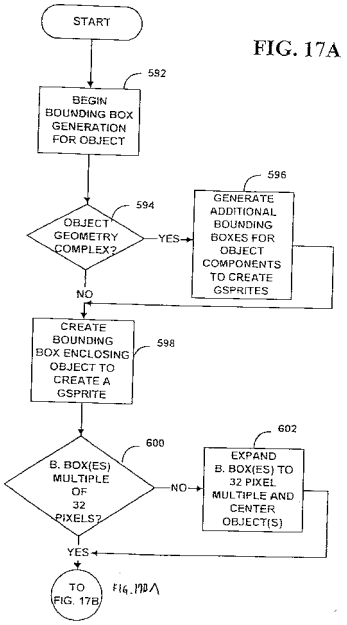


FIG. 16B



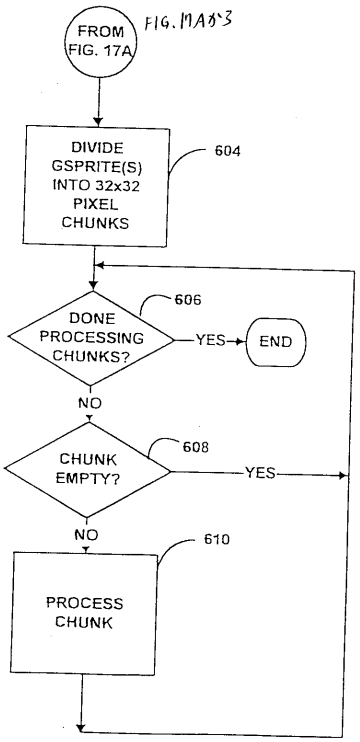
【図１７Ａ】

FIG. 17A



【図１７Ｂ】

FIG. 17B



【図１８】

FIG. 18A

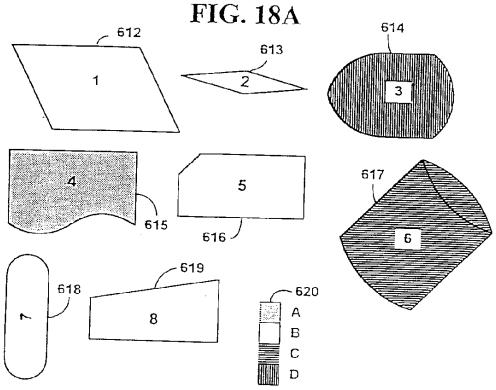
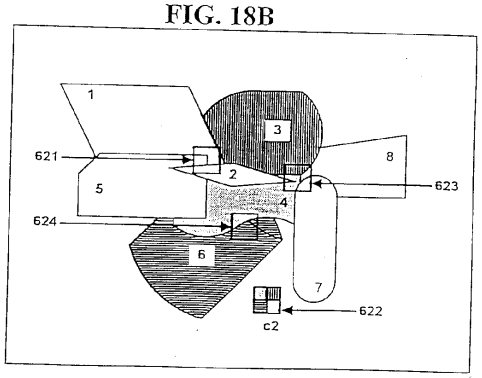


FIG. 18B



【図１９】

FIG. 19A

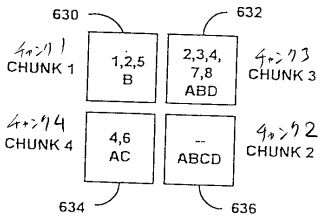
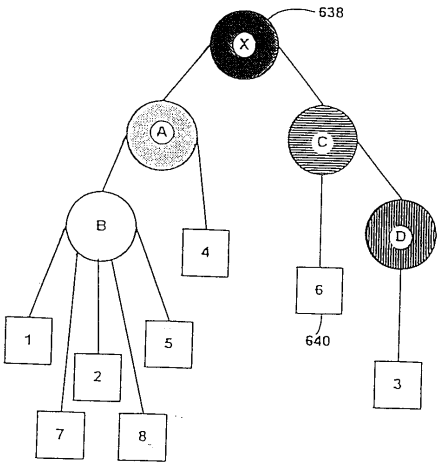
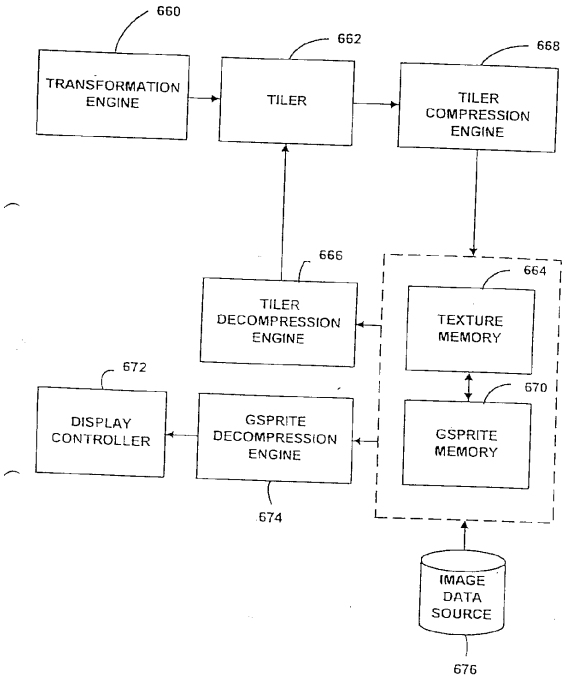


FIG. 19B



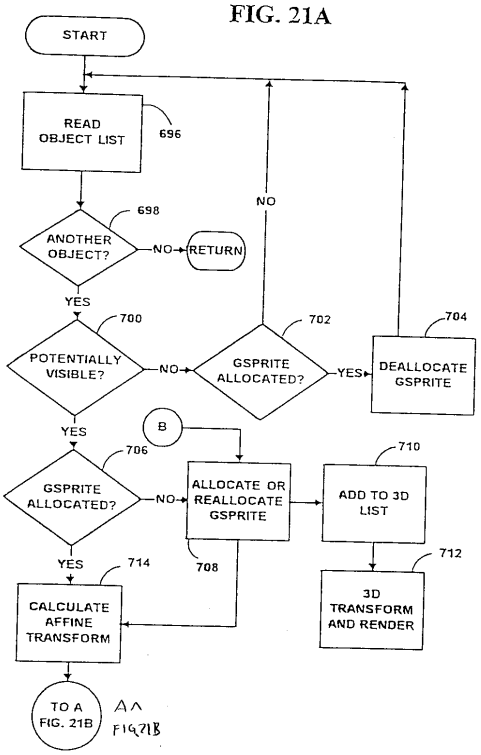
【図２０】

FIG. 20



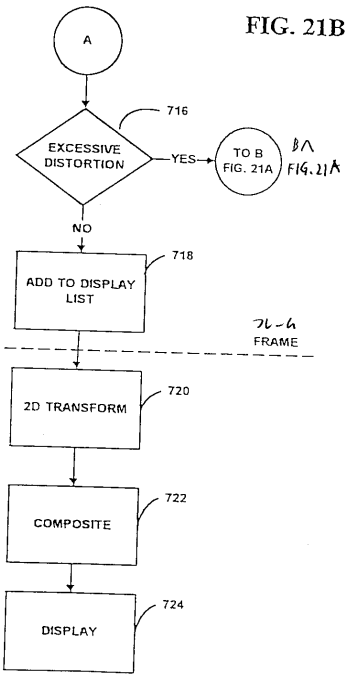
【図２１Ａ】

FIG. 21A



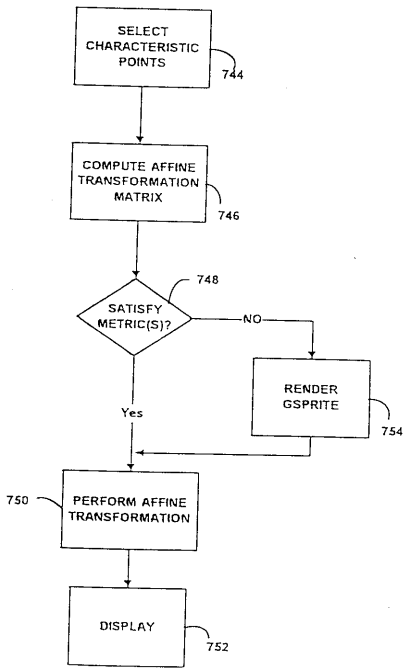
【図２１Ｂ】

FIG. 21B



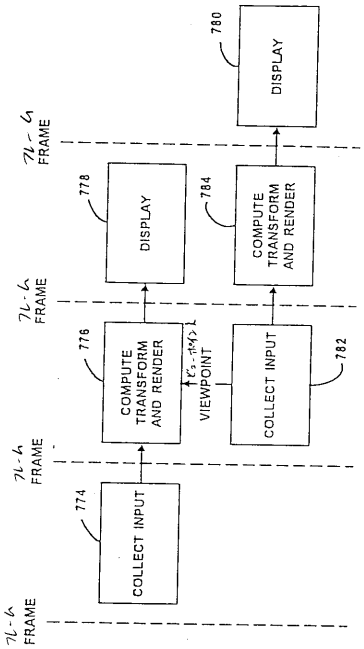
【図２２】

FIG. 22



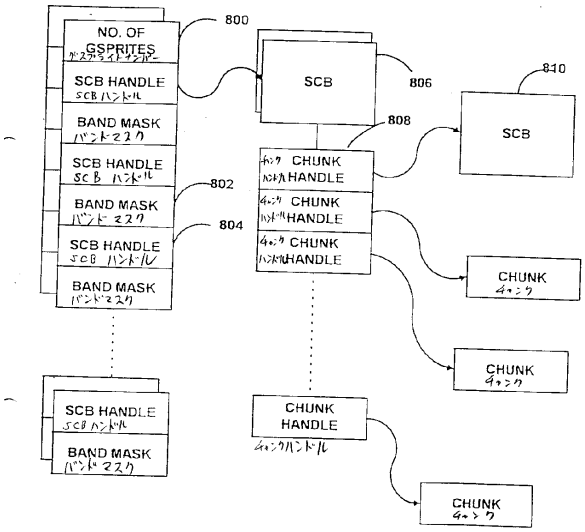
【図２３】

FIG. 23



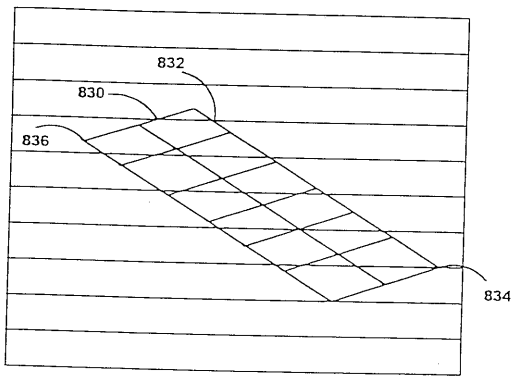
【図２４】

FIG. 24



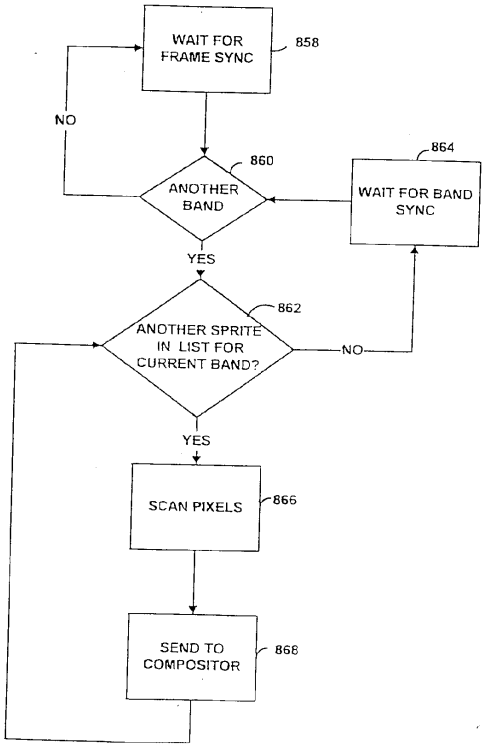
【図２５】

FIG. 25



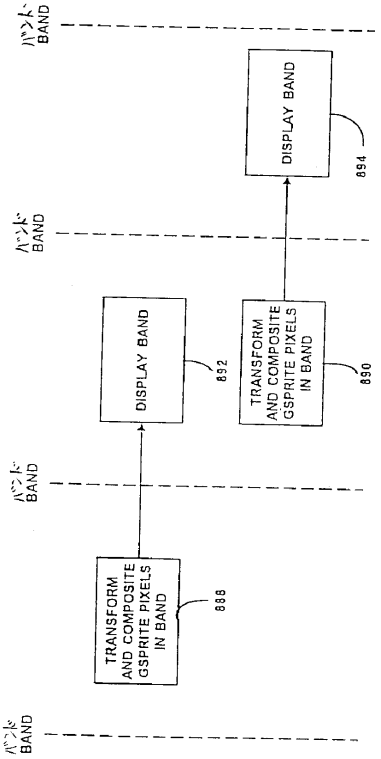
【図２６】

FIG. 26



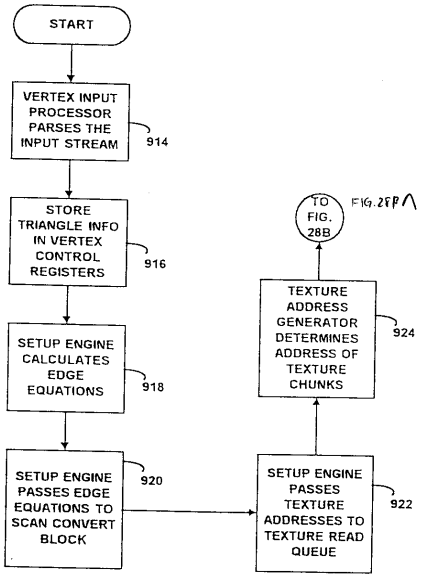
【図２７】

FIG. 27



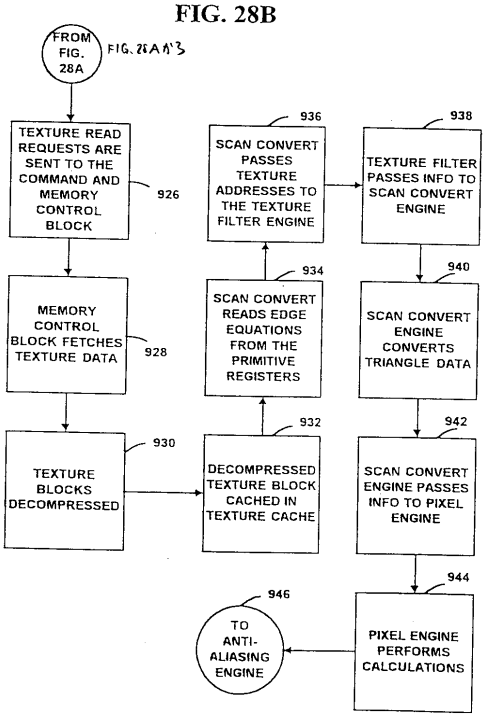
【図２８Ａ】

FIG. 28A



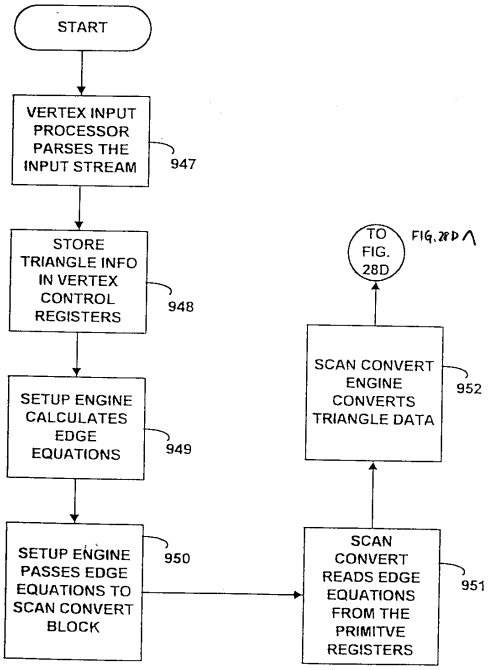
【図２８Ｂ】

FIG. 28B



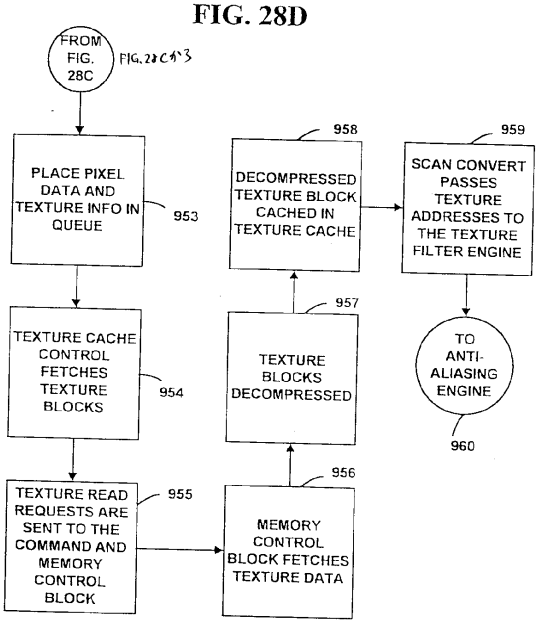
【図２８Ｃ】

FIG. 28C



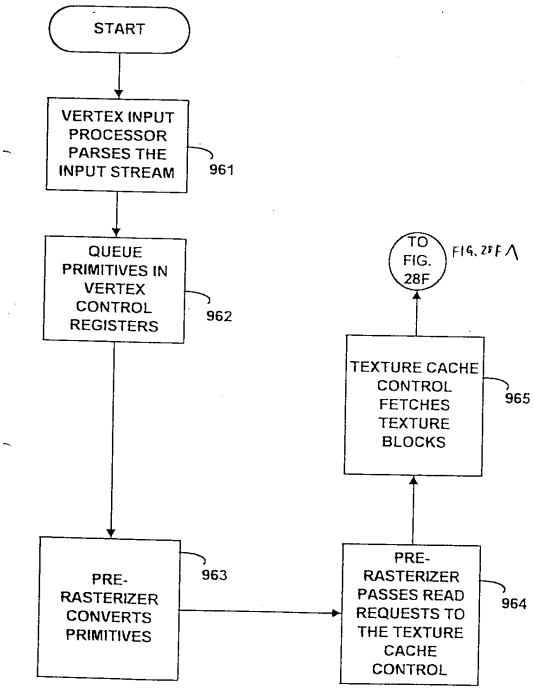
【図２８Ｄ】

FIG. 28D



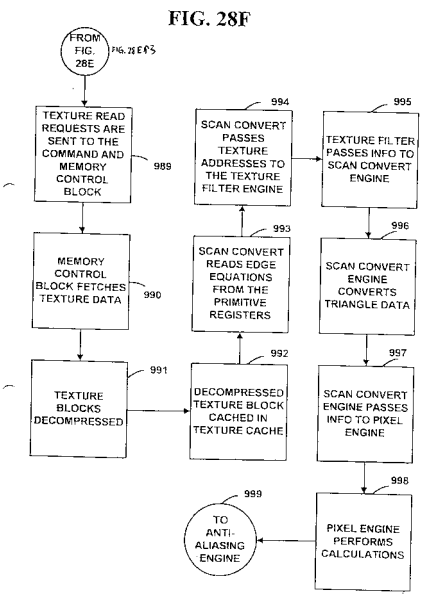
【図２８Ｅ】

FIG. 28E



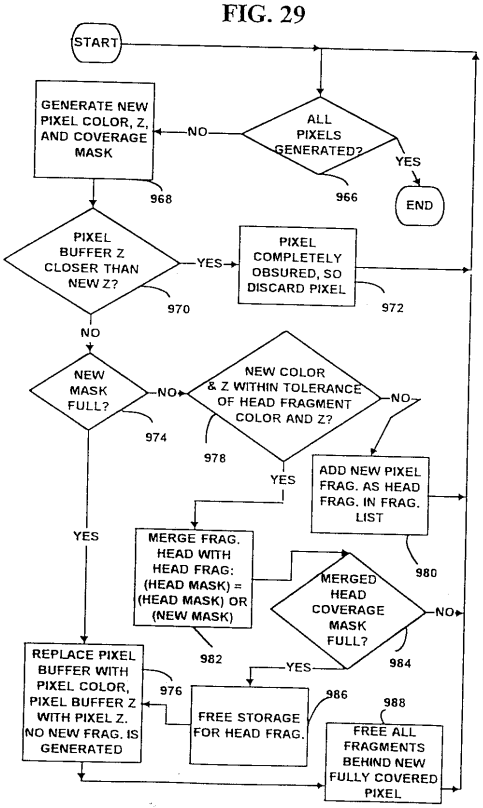
【図２８Ｆ】

FIG. 28F



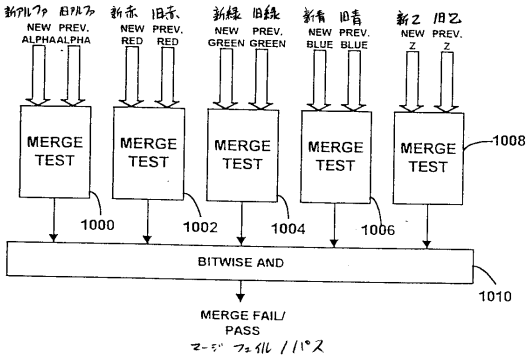
【図２９】

FIG. 29



【図３０】

FIG. 30



【図３１】

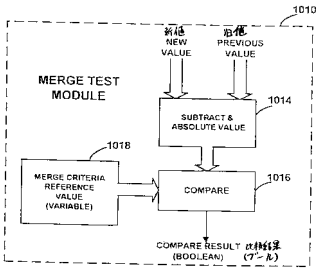
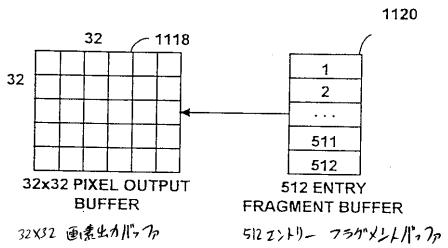


FIG. 31

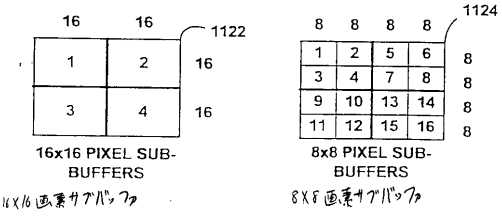
【図３２】

FIG. 32



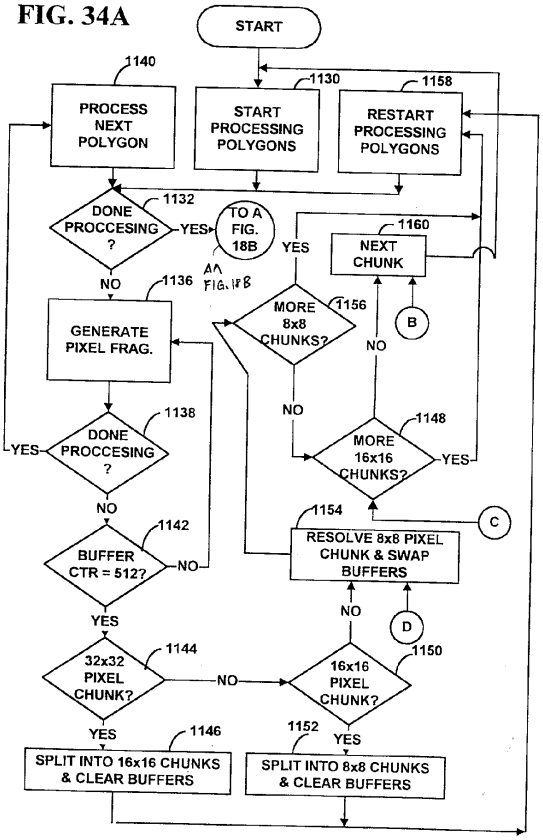
【図３３】

FIG. 33



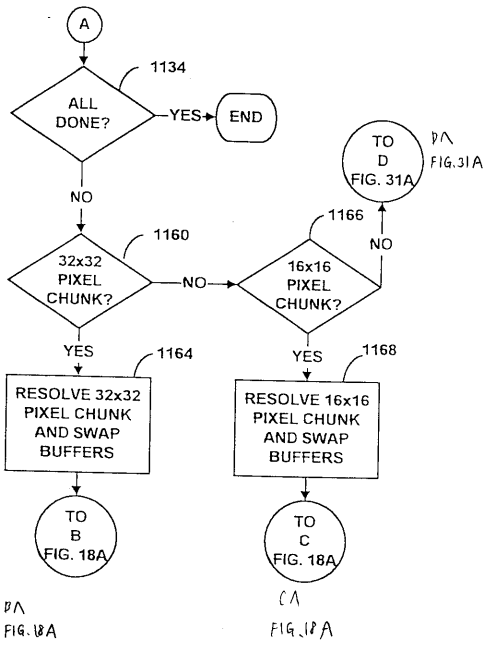
【図３４Ａ】

FIG. 34A



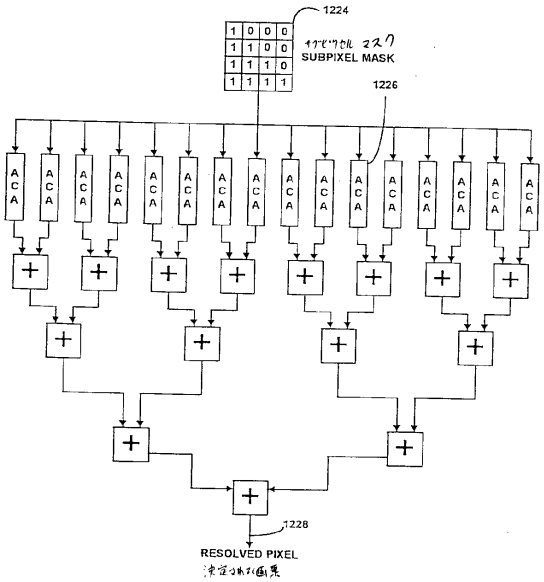
【図３４Ｂ】

FIG. 34B



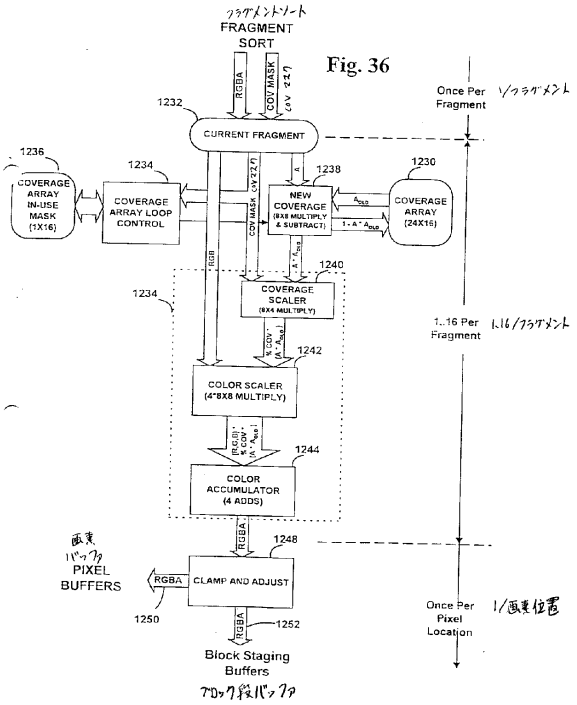
【図３５】

FIG. 35



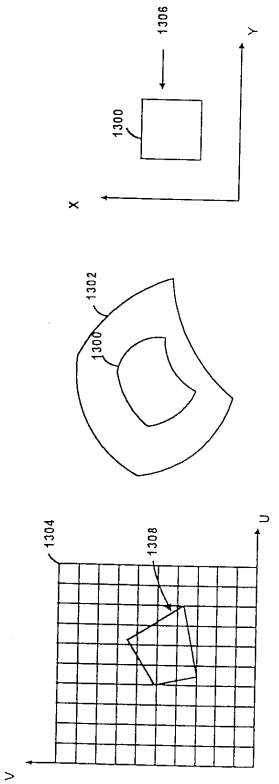
【図３６】

Fig. 36



【図３７】

FIG. 37



【図３８】

FIG. 38

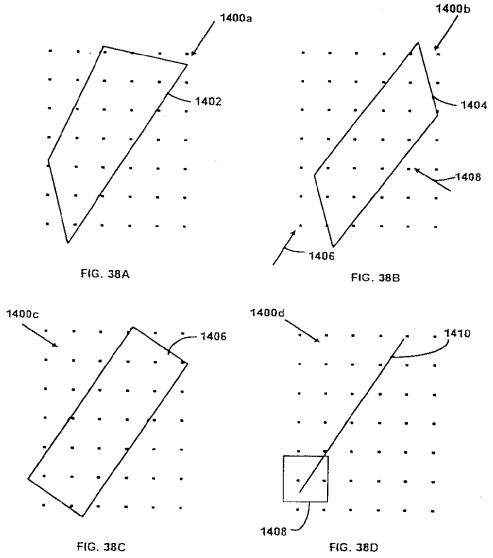
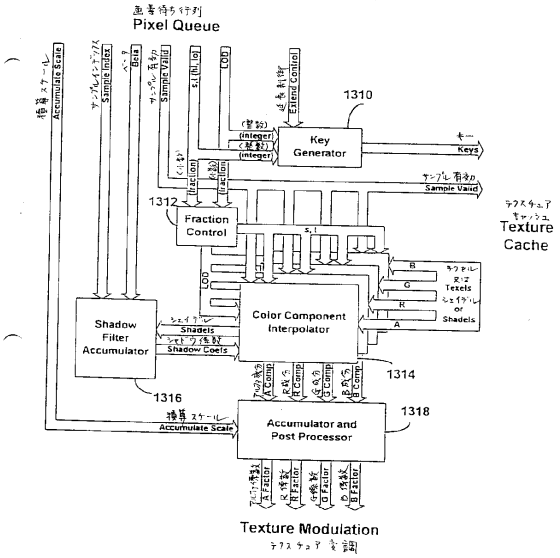


FIG. 38A FIG. 38B

FIG. 38C FIG. 38D

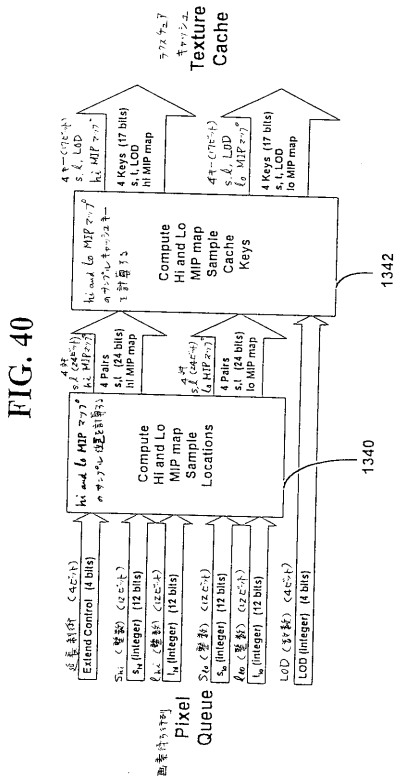
【図３９】

FIG. 39



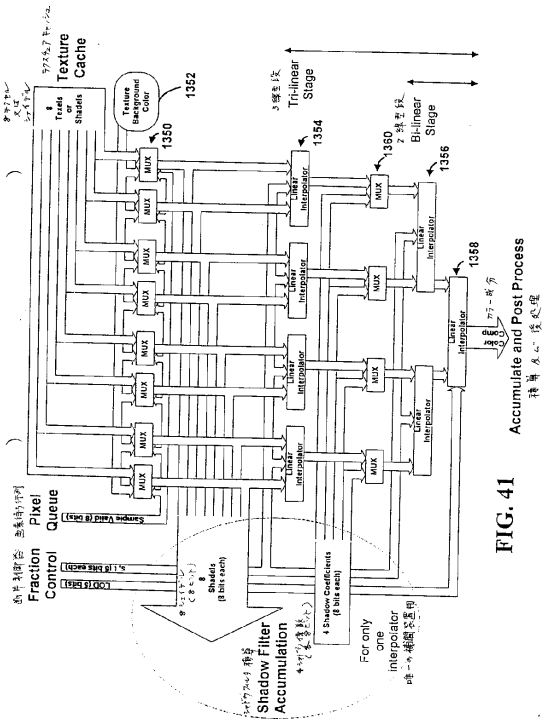
【図４０】

FIG. 40



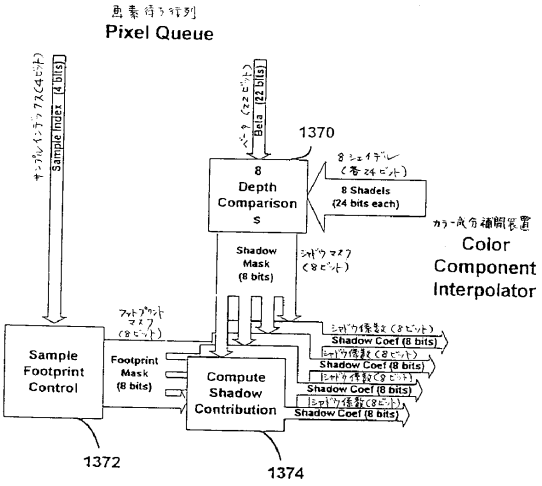
【図４１】

FIG. 41



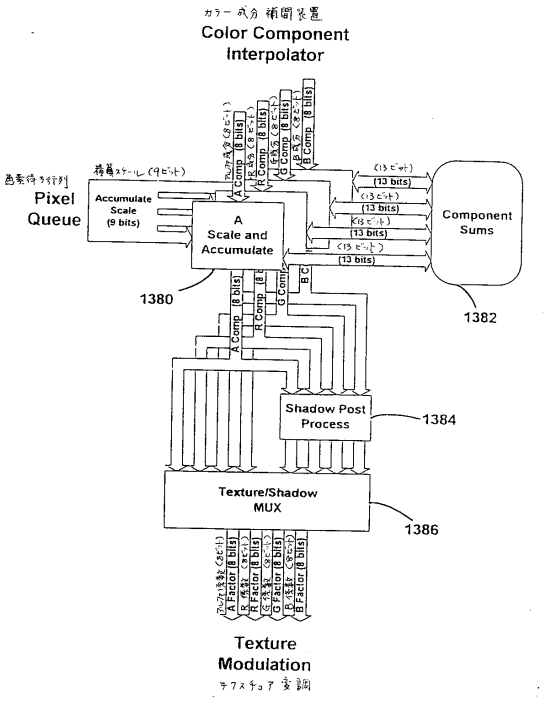
【図４２】

FIG. 42



【図４３】

FIG. 43



フロントページの続き

(31)優先権主張番号 08/671,412
(32)優先日 平成8年6月27日(1996.6.27)
(33)優先権主張国 米国(US)
(31)優先権主張番号 08/671,506
(32)優先日 平成8年6月27日(1996.6.27)
(33)優先権主張国 米国(US)
(31)優先権主張番号 08/672,347
(32)優先日 平成8年6月27日(1996.6.27)
(33)優先権主張国 米国(US)
(31)優先権主張番号 08/672,425
(32)優先日 平成8年6月27日(1996.6.27)
(33)優先権主張国 米国(US)
(31)優先権主張番号 08/672,694
(32)優先日 平成8年6月27日(1996.6.27)
(33)優先権主張国 米国(US)

(72)発明者 ミールフォルト ネイサン ピー
アメリカ合衆国 ワシントン州 ９８００５ ベールヴ ワンハンドレッド サーティフォース アヴェニュー エヌ イー ３４４１
(72)発明者 カジャ ジェイムス ティー
アメリカ合衆国 ワシントン州 ９８０１９ デュヴァル エヌ イー ビッグ ロック ロード ３１４３３
(72)発明者 トルボーグ ジョン ジー ジュニア
アメリカ合衆国 ワシントン州 ９８０５２ レドマンド エヌ イー フィフティース ウェイ １６４０７
(72)発明者 ケンワーシー マーク エル
アメリカ合衆国 ワシントン州 ９８０１９－７８０６ デュヴァル エヌ イー ワンハンドレッド セヴンティー セカンド ストリート ３０３３０
(72)発明者 トール マイケル アレン
アメリカ合衆国 ワシントン州 ９８００８ ベールヴ ワンハンドレッド シックスティース プレイス エヌ イー ２８９８
(72)発明者 グリフィン ケント イー
アメリカ合衆国 ワシントン州 ９８００８ ベールヴ ワンハンドレッド シックスティセヴンス プレイス エヌ イー ２５１１
(72)発明者 レングイェル ジェローム エドワード
アメリカ合衆国 ワシントン州 ９８１０８ シアトル エヌ フォーティセカンド ストリート １８０８
(72)発明者 ガブリエル スティーヴン エイ
アメリカ合衆国 ワシントン州 ９８０５２－３２５６ レドマンド エヌ イー ナインティース ストリート １７７５６ アパートメント エヌ－３７２
(72)発明者 ヴェルス ジェイムス イー
アメリカ合衆国 ワシントン州 ９８０７２ ウッディン ヴィル ワンハンドレッド セヴンティファースト レイン エヌ イー １４０２９
(72)発明者 チャウヴィン ジョセフ ダブリュー
アメリカ合衆国 ワシントン州 ９８０２９ イサッカ エス イー サーティエイス プレイス ２４２０５
(72)発明者 グッド ハワード
アメリカ合衆国 ワシントン州 ９８１０５ シアトル エヌ イー フォーティサード ストリ

ート　７１１
(72)発明者　パウエル　ウィリアム　チャンバース　ザ　サード
アメリカ合衆国　ワシントン州　９８１２２　シアトル　シックスティーンス　アヴェニュー　８１７

合議体
審判長　吉村　博之
審判官　千葉　輝久
審判官　加藤　恵一

(56)参考文献　特開平６－２３６４２９（ＪＰ，Ａ）
特開平７－１４１５２０（ＪＰ，Ａ）
特開平５－２３３７７９（ＪＰ，Ａ）
国際公開第９５／０２２３６（WO，Ａ1）
特開平６－３６０１１（ＪＰ，Ａ）
特開平７－１１０８７３（ＪＰ，Ａ）