(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4027463号
(P4027463)
(45)発行日　平成19年12月26日(2007.12.26)　(24)登録日　平成19年10月19日(2007.10.19)

(51)Int.Cl.　　F I
G06T 11/20 (2006.01)　　G06T 11/20 100
G06T 15/00 (2006.01)　　G06T 15/00 100A

請求項の数 12　(全 42 頁)

(21)出願番号　特願平9-167040
(22)出願日　平成9年6月24日(1997.6.24)
(65)公開番号　特開平10-111948
(43)公開日　平成10年4月28日(1998.4.28)
審査請求日　平成16年6月21日(2004.6.21)
(31)優先権主張番号　690,432
(32)優先日　平成8年7月26日(1996.7.26)
(33)優先権主張国　米国(US)

(73)特許権者　398038580
ヒューレット・パッカード・カンパニー
HEWLETT-PACKARD COMPANY
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　３０００
(74)代理人　100081721
弁理士　岡田　次生
(72)発明者　ブラッドリー・エル・サンダース
アメリカ合衆国８０５２６コロラド州フォート・コリンズ、ラトリッジ・コート　１８０１

審査官　橋爪　正樹

最終頁に続く

(54)【発明の名称】テクセル・データを記憶する方法およびテクセル・データを記憶するシステム

(57)【特許請求の範囲】
【請求項１】
コンピュータ・グラフィックス・システムにおいてテクスチャ・マッピング・ハードウェアの連続するメモリブロックにテクセル・データを記憶する方法であって、
テクスチャ・マッピング・ハードウェアによって、テクスチャマップの第１のレベルに対応する第１のデータセットを受け取り、
テクスチャ・マッピング・ハードウェアによって、前記第１のデータセットを分析して前記第１のデータセットのサイズおよびレベル番号を求め、
テクスチャ・マッピング・ハードウェアによって、テクスチャマップの第２のレベルに対応する第２のデータセットを受け取り、
テクスチャ・マッピング・ハードウェアによって、前記第２のデータセットを分析して前記第２のデータセットのサイズおよびレベル番号を求め、
テクスチャ・マッピング・ハードウェアによって、前記第２のデータセットの前記サイズおよび前記レベル番号と、前記第１のデータセットの前記サイズおよび前記レベル番号とを比較し、
テクスチャ・マッピング・ハードウェアによって、前記比較に基づいて、前記第２のデータセットが、前記第１のデータセットと整合しているか否かを判定し、
テクスチャ・マッピング・ハードウェアによって、前記判定において前記第２のデータセットが前記第１のデータセットと整合すると判定されたとき、前記連続するメモリブロックにおける前記第２のデータセットの前記レベル番号に基づく位置に前記第２のデータ

セットを記憶する、
ことを含む連続するメモリブロックにテクセル・データを記憶する方法。
【請求項２】
前記判定するステップにおいて前記第２のレベルが前記第１のデータセットと整合しないと判定されたとき、前記第２のデータセットを廃棄することをさらに含む、請求項１記載のテクセル・データを記憶する方法。
【請求項３】
前記判定するステップにおいて前記第２のデータセットが前記第１のデータセットと整合しないと判定されたとき、前記第２のデータセットをメモリに記憶することをさらに含み、前記メモリは前記連続するメモリブロックの外にある、請求項１記載のテクセル・データを記憶する方法。
【請求項４】
前記第１のデータセットの前記サイズおよび前記レベル番号に基づいて、前記テクスチャマップを記憶するのに十分なメモリ量を求め、
少なくとも前記テクスチャマップを記憶するのに十分な前記メモリ量の前記連続するメモリブロックを割り当てること、
をさらに含む請求項１記載のテクセル・データを記憶する方法。
【請求項５】
前記第１のデータセットを廃棄することをさらに含む、請求項３記載のテクセル・データを記憶する方法。
【請求項６】
前記第１のデータセットの前記サイズおよび前記レベル番号に基づいて、前記テクスチャマップのレベル番号を求めることをさらに含む、請求項４記載のテクセル・データを記憶する方法。
【請求項７】
テクセル・データをコンピュータグラフィックシステムの連続するメモリブロックに記憶するシステムであって、
テクスチャマップの第１のレベルに対応する第１のデータセットを受け取る手段と、
前記第１のデータセットを分析して前記第１のデータセットのサイズおよびレベル番号を求める手段と、
テクスチャマップの第２のレベルに対応する第２のデータセットを受け取る手段と、
前記第２のデータセットを分析して前記第２のデータセットのサイズおよびレベル番号を求める手段と、
前記第１のデータセットの前記サイズおよび前記レベル番号と前記第２のデータセットの前記サイズおよび前記レベル番号とを比較する手段と、
前記比較する手段に基づいて、前記第２のデータセットが前記第１のデータセットと整合するか否かを判定する手段と、
前記判定する手段によって前記第２のデータセットが前記第１のデータセットと整合すると判定されたとき、前記連続するメモリブロックにおける前記第２のメモリブロックの前記レベル番号に基づく位置に前記第２のデータセットを記憶する手段と、
を備える、前記システム。
【請求項８】
前記判定する手段によって、前記第２のデータセットが前記第１のデータセットと整合しないと判定されたとき、前記第２のデータセットを廃棄する手段をさらに備える請求項７記載のシステム。
【請求項９】
前記判定する手段によって、前記第２のデータセットが前記第１のデータセットと整合しないと判定されたとき、前記第２のデータセットをメモリへ記憶する手段をさらに備え、前記メモリは前記連続するメモリブロックの外部である、請求項７記載のシステム。
【請求項１０】

前記第１のデータセットの前記サイズおよび前記レベル番号に基づいて前記テクスチャマップを記憶するのに十分なメモリ量を求める手段と、
少なくとも前記テクスチャマップを記憶するのに十分な前記メモリ量の前記連続するメモリブロックを割り当てる手段と、
を備える、請求項７記載のシステム。
【請求項１１】
前記第１のデータセットを廃棄する手段をさらに備える、請求項９記載のシステム。
【請求項１２】
前記第１のデータセットの前記サイズおよび前記レベル番号に基づいて前記テクスチャマップのレベル番号を求める手段をさらに備える、請求項１０記載のシステム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は、コンピュータ・グラフィックス・システムにおけるキャッシュの一貫性を維持する方法に関するもので、特に、テクスチャ・マッピング・コンピュータ・グラフィックス・システムのテクスチャ・マップのソフトウェア・メモリ管理に関するものである。
【０００２】
【従来の技術】
テクスチャ・マッピングの従来技術は、アメリカ合衆国特許出願第08/486,447号に記載されているが、その内容は、ユーザのテクスチャのコピーをテクスチャ照会メカニズムを提供するソフトウェアに記憶し、一度にすべてのテクセルを収納することのできる十分なメモリがない場合でもハードウェア上のテクセルのキャッシュの実行を可能にさせる技術に関するものである。
【０００３】
典型的なコンピュータ・グラフィックス・システムにおいては、表示画面上に表されるべきオブジェクトは、複数のグラフィックス・プリミティブに分解される。プリミティブは、グラフィックス・ピクチャの基本コンポーネントであって、点、線、ベクトルおよび三角形のような多角形を含む場合がある。典型的には、ハードウェア／ソフトウェア方式の実施によって、画面上に表される１つまたは複数のオブジェクトの像を表すグラフィックス・プリミティブが、２次元表示画面上に描画(すなわちレンダリング)される。
【０００４】
描画されるべき３次元オブジェクトを定義するプリミティブは、典型的には、プリミティブ・データとして各プリミティブを定義するホスト・コンピュータから提供される。例えば、プリミティブが三角形である場合、ホスト・コンピュータは、各頂点をｘ，ｙ，ｚ座標およびＲ，Ｇ，Ｂカラー値を使用して表すことによってそのプリミティブを定義する。各プリミティブを表すためオンにセットされるべき表示画面ピクセルおよび各ピクセルに関するＲ，Ｇ，Ｂ値を計算する際に、レンダリング・ハードウェアはプリミティブ・データを補間する。
【０００５】
初期のグラフィックス・システムは、画像表示のため、複雑な３次元オブジェクトを表現またはモデル化できる十分現実的な方法を持つことができなかった。そのようなシステムによって画面表示される画像は、円滑な表面の欠如したテクスチャ、でっぱり、ひっかき傷、陰影またはその他モデル化されているオブジェクトに存在する表面細部を露出させた。その結果、表面細部を改善した画像を表示するため種々の方法が開発された。テクスチャ・マッピングは、そのような方法の１つである。この方法は、ソース画像(本明細書では「テクスチャ」と呼ぶ)を３次元オブジェクト表面上へマップし、その後その３次元オブジェクトを２次元グラフィックス表示画面へマップすることによって、画像表示を行う。一般的にはマップされたテクスチャである表面細部属性は、カラー、鏡面反射、ベクトル摂動、反射、透明度、影、表面不規則および階調を含む。
【０００６】

テクスチャ・マッピングは、１つまたは複数の点からなるテクスチャ・エレメント(すなわち「テクセル」)を、テクスチャをマップする対象であるオブジェクトの表示部分の各点エレメント(すなわち「ピクセル」)に当てはめることを伴う。テクスチャ・マッピング・ハードウェアは、テクスチャ・マップにおけるテクセルがオブジェクトを表現する表示画面上のピクセルにどのように対応するかを示す情報を伝統的に備えている。テクスチャ・マップにおける各テクセルは、２次元テクスチャ・マップにおけるその位置を識別するＳおよびＴ座標によって定義される。各ピクセル毎に、それにマップする対応する１つまたは複数のテクセルが、テクスチャ・マップから取り出され、表示画面上でテクスチャ・オブジェクトを表現するピクセルのために生成される最終的Ｒ，Ｇ，Ｂ値に組み込まれる。

【０００７】

オブジェクト・プリミティブの各ピクセルがテクスチャ・マップにおける単一のテクセルと必ずしも一対一で対応しない点は注意されるべきである。例えば、オブジェクトが表示画面に表される視認ポートに接近すればする程、オブジェクトは大きく拡大される。オブジェクトが表示画面上で拡大されるにつれ、テクスチャの表示は次第に詳細になる。従って、オブジェクトが表示画面のかなりの部分を消費する時、そのオブジェクトを表示画面で表すため多数のピクセルが使用され、オブジェクトを表す各ピクセルがテクスチャ・マップにおける単一のテクセルと一対一の対応関係でマップ(対応づけ)するか、あるいは単一のテクセルが複数のピクセルにマップする。しかし、オブジェクトが表示画面の比較的小さい部分を占める時、オブジェクトを表すため比較的少数のピクセルが使用され、その結果、テクスチャの表現は粗くなり、各ピクセルが複数のテクセルにマップする。更に、１つのテクスチャがオブジェクトの小さい部分にマップされる時、各ピクセルは複数のテクセルにマップするかもしれない。複数のテクセルにマップするピクセルの各々毎に、目的のテクセル・データが計算される。１つのピクセルが複数のテクセルにマップするのが一般的であるので、１つのピクセルに対応するテクセル・データは、典型的には、そのピクセルにマップする複数テクセルの平均を表現する。

【０００８】

テクスチャ・マッピング・ハードウェア・システムは、典型的には、レンダリングされているオブジェクトに関するテクスチャを表現するデータを記憶するローカル・メモリを含む。上述の通り、１つのピクセルは複数のテクセルに対応する。テクスチャ・マッピング・ハードウェアが１つのピクセルに対応する多数のテクセルをローカル・メモリから読み取って平均値を生成しなければならないとすれば、多数のメモリ読取りと多くのテクセル値の平均算出が必要となり、そのため、処理時間の浪費とシステム性能の低下を生む。

【０００９】

この問題を克服するため、各テクスチャに関して「ＭＩＰ」マップと呼ばれる一連のマップを作成してこれを利用する方式が開発された(ＭＩＰは、狭い場所に多数のものを意味する"multum in parvo"の頭文字をとったものである)。この方式は、レンダリングされつつあるオブジェクトに関連するテクスチャの一連のＭＩＰマップをテクスチャ・マッピング・ハードウェアのローカル・メモリに記憶する。あるテクスチャに関するＭＩＰマップは、テクスチャ・マップに直接対応する基底マップ(すなわち「レベル０」マップ)および一連の後続のマップを含む。後続マップの各々のサイズは、テクスチャ・マップの各次元において、先行マップに対して２単位減少する。図１に、１つのＭＩＰマップ・セットの例が示されている。このＭＩＰマップは、サイズが８×８テクセルである(レベル０の)基底マップ１００と共に、サイズ的にそれぞれ４×４テクセルであるレベル１、２×２テクセルであるレベル２および１×１テクセルであるレベル３を表す一連のマップ１０２、１０４および１０８を含む。

【００１０】

レベル１の４×４マップ１０２は、レベル１マップ１０２の中の各テクセルがレベル０基底マップ１００の４つのテクセルの平均に対応するように、基底マップ１００をボックス・フィルタリング(すなわち比例縮小)することによって生成される。例えば、レベル１マ

ップ１０２のテクセル１１０は、レベル０基底マップ１００のテクセル１１２－１１５の平均に等しい。同様に、レベル１マップ１０２のテクセル１１８および１２０は、それぞれレベル０基底マップ１００のテクセル１２１－１２４および１２５－１２８の平均に等しい。同様に、レベル２の２×２マップ１０４は、レベル２マップ１０４の中のテクセル１３０がレベル１マップ１０２のテクセル１１０および１１８－１２０の平均に等しくなるように、レベル１マップ１０２をボックス・フィルタリングすることによって生成される。レベル３マップ１０８における単一テクセルは、レベル２マップ１０４における４つのテクセルを平均して生成される。

【００１１】

従来技術のグラフィックス・システムは、一般的に、表示画面上に描画されるべきプリミティブに関して使用されるはずのすべてのテクスチャのための完全なＭＩＰマップ・セットを、ホスト・コンピュータのメイン・メモリからテクスチャ・マッピング・ハードウェアのローカル・メモリへ、ダウンロードする。当業者には理解されることであろうが、完全なＭＩＰマップ・セットとは、レベル０からレベルＮまでの(但しレベルＮは１×１ＭＩＰマップ)すべてのＭＩＰマップを意味する。このようにして、テクスチャ・マッピング・ハードウェアは、一連のＭＩＰマップのどのレベルのマップからでもテクスチャ・データを取り出すことができる。特定のピクセルに関してテクセル・データを取り出すためどのマップにアクセスすべきかの決定は、ピクセルがマップするテクセルの数に基づく。例えば、ピクセルが、テクスチャ・マップにおける単一のテクセルと一対一の対応関係でマップする場合、基底マップ１００がアクセスされる。しかしながらピクセルが４、１６または６４のテクセルへマップするとすれば、マップ１０２、１０４および１０８がそれぞれアクセスされる。なぜならば、それらのマップは、テクスチャ・マップにおける４、１６および６４のテクセルの平均を表現しているテクセル・データをそれぞれ記憶しているからである。

【００１２】

理解される通り、一連のテクスチャＭＩＰマップは記憶のため多量のシステム・ソフトウェア・メモリを必要とする。例えば、１０２４×１０２４のテクセルのテクスチャ基底マップを持つテクスチャのための一連のＭＩＰマップは、ＭＩＰマップされたテクスチャの１コピーを記憶するため５メガバイト以上のシステム・ソフトウェア・メモリを必要とする。従って、ＭＩＰマップされたテクスチャの複数コピーの記憶は、膨大な量のシステム・ソフトウェア・メモリを使用する。

【００１３】

システム・ソフトウェア・メモリは、数ギガバイトまでのソフトウェア・データを記憶することができるが、考慮されなければならないもう一つの問題は、ＭＩＰマップを実際にどこへ記憶するかである。特に、グラフィックス画像の高速描画を達成するためには、テクセル情報を適切なレベルのＭＩＰマップからグラフィックス・ディスプレイへ可能な限り迅速に転送することができることが重要である。すべてのレベルの位置が前もってわかっていれば最善ではあるが、典型的なグラフィックス適用業務プログラム・インターフェース(すなわちＡＰＩ)稼働形態はそのタスクを非常に困難にさせる。特に、ヒューレット・パッカード社から提供されている"OpenGL"と呼ばれるグラフィックスＡＰＩは、ユーザがＭＩＰマップをダウンロードする際種々のレベルをどのようなレベルの順序でもメモリに送ることを可能にする。

【００１４】

【発明が解決しようとする課題】

現在までは、オペレーティング・システムのメモリ割り当てルーチンによって要求元に返されるメモリを各レベル毎の記憶位置として使用して、そのようなＭＩＰマップ・レベルを個別にメモリに記憶する形態がとられて来た。このように、ＭＩＰマップのレベルが記憶されるメモリの実際の位置は、オペレーティング・システムに委ねられた。従って、個々のレベルが必要とされる時点で記憶位置を特定しなければならず、そのためシステム動作速度の一般的低下をもたらしている。従って、複数のレベルのＭＩＰマップを整合性の

ある形態でメモリに記憶することが必要とされている。

【００１５】

【課題を解決するための手段】

本発明は、上記課題を解決するため、データの完全性を維持しながら、ＭＩＰ化されたOpenGLテクスチャ・マップの複数レベルを連続的メモリに記憶することを可能にするメカニズムを提供する。それによって、本発明は、完全にＭＩＰ化されたテクスチャ・マップをハードウェアにダウンロードする際あるいはテクスチャ・マッピングがソフトウェア・ラスタライゼーションを使用する際に発生する可能性のあるメモリ・キャッシュ・ミスを減少あるいは除去する。また、たとえデータが現在時の完全ＭＩＰマップ記述に適合しない場合でも、本発明は、ダウンロードされつつあるデータの完全性を維持する。

【００１６】

本発明のアルゴリズムは、コンピュータ・グラフィックス・システムにおける連続的メモリにテクセル・データをダウンロードする方法を提供する。該方法は、ダウンロードされる最初のテクセル・データのＭＩＰマップのレベル番号およびサイズを基にして、当該テクスチャに関連するすべてのＭＩＰマップに必要な連続的メモリ量を割り当て、上記最初のＭＩＰマップに関するデータを、レベル番号およびサイズを基にして計算するオフセット値に従ってそのレベル番号にとって適切な上記連続的メモリ位置に記憶する。更に、本発明のアルゴリズムは、いくつかの後続のＭＩＰマップがダウンロードされるにつれて、それらの各々が上記最初にダウンロードされたＭＩＰマップとレベル番号およびサイズの観点から整合しているか否かを判断し、上記判断の結果整合していれば、それらレベル番号に基づいて計算されるオフセット値が示す上記連続的メモリ位置へそれらレベルのデータを記憶し、上記判断の結果整合していない場合、それらレベルのデータを一時的メモリへ記憶する。上記後続のＭＩＰマップにおいて基底レベルのデータがダウンロードされる毎に、上記一時的メモリに記憶されたデータを検査して、当該基底レベルと整合性があるレベルのデータがあれば、それらを当該基底レベルが記憶される連続的メモリへ移動する。

【００１７】

【発明の実施の形態】

本発明に従って、図２ないし図６に示される本発明の方法１０は、テクスチャ・データを連続的メモリに記憶することを目的とする。上述の通り、テクスチャ・データは、レベル０からレベルＮまでのすべてのレベルに関連するデータから構成される(但しレベルＮは１×１配列である)。先ず第１に、すべてのテクセル・データを単一メモリ・ブロックに記憶できる十分なメモリが存在するか否かを決定しなければならない。この決定は図２のブロック１４で行われる。十分なメモリがあるかどうか判明しない場合、必要とされる連続メモリ・ブロックのサイズを計算し(ブロック１６)、そのメモリを割り当て(ブロック１８)なければならない。例としてレベル０ＭＩＰマップが８×８マップであると仮定すれば、それは６４個の「位置」を占有するが、実際のメモリ量はテクセルあたりのバイト数(×６４)によって決定される。この場合、レベル１ＭＩＰマップは４×４すなわち１６の位置を占め、レベル２ＭＩＰマップは２×２すなわち４つの位置を占め、レベル３ＭＩＰマップは１×１すなわち１つの位置を占める。かくして、位置の総数は８５で、メモリ量はテクセルあたりのバイト数×８５である。レベル２ＭＩＰマップが２×２マップであるということは基底マップが８×８マップであることを意味する点は容易に理解されるであろう。従って、ＭＩＰマップのレベルとサイズを所与とすれば、完全なＭＩＰマップを記憶するために必要な連続メモリ・ブロックのサイズは容易に決定することができる。

【００１８】

十分なメモリを割り当てることができないと判断すれば(ブロック２０)、エラー条件が発生する(ブロック２２)。そうではなく十分なメモリを割り当てることができると判断すれば(ブロック２０)、基底マップ(レベル０)が記憶される(ブロック２４)。次に、本発明の好ましい実施形態では、"Level0K"と呼ばれるフラグが０にセットされ(ブロック２６)、レベル情報がＯＫか否かが判断される(ブロック２８)。これは、ロードされつつあるＭＩ

Ｐマップ・レベルに関する情報が、ＭＩＰマップに関する既知の情報と整合しているか否かに関して判断することを意味する。例を用いて説明すれば、４×４のサイズを持つ「レベル１」ＭＩＰマップがダウンロードされ、次に８×８のサイズを持つ「レベル０」ＭＩＰマップがダウンロードされれば、データは整合していて、レベルはＯＫである。これとは相違して、４×４のサイズを持つ「レベル１」ＭＩＰマップがダウンロードされ、次に４×４のサイズを持つ「レベル０」ＭＩＰマップがダウンロードされれば、データは整合してなく、レベルはＯＫではない。データが以前にダウンロードされたデータと整合していれば、LevelOKフラグは「１」にセットされる。

【００１９】

LevelOKが「０」であれば、そのＭＩＰマップが基底マップ(レベル０)に関するものであったか否かが判断される(ブロック３２)。もしもそうではなく、あるいはLevelOKが「１」に等しかったならば、処理の流れは図５の"C"へ進む。そのＭＩＰマップが基底マップ(レベル０)に関するもので、LevelOKフラグが「１」にセットされていれば(ブロック３４)、図３のブロック３８(記号"A")へ進み、基底レベル情報および既存メモリへのポインタを記憶し(ブロック５０および４０)、次に、連続メモリのサイズが計算され(ブロック４２)、連続メモリが割り当てられる(ブロック４４)。連続メモリが正しく割り当てられたか否かを検証し(ブロック４６)、エラーがあったと判断すれば、それ以上の動作は行われない。

【００２０】

次に、後続のループにおいて繰り返し動作を行うため、カウンタ、増分式および検証プロシージャ５６が設定される(ブロック５２、５４および５６)。本発明の好ましい実施形態において、ループは、現在時レベルへのポインタを入手するステップ(ブロック６２)、および現在時レベルがまだメモリ６４のいかなる位置をもポイントしていないことを判断するステップ(ブロック６４)を含む。このループは繰り返され、繰り返し毎にループ・カウンタが増分され(ブロック５４)、潜在的にサポートされているレベルの最大数に対応する回数に達するまでループは繰り返される。本発明の実施形態において、現行使用されるハードウェアおよびメモリ容量に基づけば、１６を越えるレベルはサポートされない。従って、ループ処理は最高１５回であるが、将来のハードウェアおよびメモリが一層大きいＭＩＰマップ・レベルの使用を示せば、この回数は増加する点は当業者に理解されることであろう。

【００２１】

現在時レベルがメモリをポイントしていると判断されれば(ブロック６４)、現在時レベルのサイズが計算され(ブロック７０)、LevelOKが「０」にセットされて、データが正しくないことが示される(ブロック７２)。次に、正確性についてレベル・データが検査され(ブロック７４)、レベル・データが正確であると判断されれば、LevelOKは、レベル・データが正確であることを示す「１」にセットされ(ブロック８８)、そのレベルに関して、基底マップの連続メモリへのオフセットが計算される(ブロック９０)。次に、現在時レベルのテクセルが連続メモリの適切な位置にコピーされる(ブロック９２)。

【００２２】

この時点において、ポインタ・フラグがセットされているか否かを検査して(ブロック９４)、セットされていなければ、図３の上記ループへ記号"E"の位置から再度入る。一方、ポインタ・フラグが「１」にセットされていれば、このレベルに関するメモリを解放し(ブロック９６)、ポインタ・フラグを「０」にセットし(ブロック９７)、現在時レベルのメモリ・オフセットを記憶する(ブロック９８)。次に、図３の上記ループへ記号"E"の位置から再度入る。

【００２３】

ブロック７４でレベル・データが正しくないと判断され、ポインタ・フラグが「１」にセットされていれば(ブロック７６)、図３の上記ループへ記号"E"の位置から再度入る。一方、ブロック７４でレベル・データが正しくないと判断され、ポインタ・フラグが「０」にセットされていれば(ブロック７６)、ポインタ・フラグを「１」にセットし(ブロック

７８)、このレベルのため一時的記憶ブロックを割り当て(ブロック８０)、メモリが正しく割り当てられたことを確認する(ブロック８２)。メモリ割り当ての際エラーが発生すればこれ以上の動作は行われない(ブロック８４)。メモリが割り当てられれば、このレベルのデータが一時メモリにコピーされ(ブロック８６)、図３の上記ループへ記号"E"の位置から再度入る。　図２に戻って、ブロック３２において、レベルが基底ブロックでない場合、処理は図５の記号"C"へ進む。また、基底レベルであるが、Level0Kフラグが「０」にセットされていなければ(ブロック３４)、処理は同様に図５の記号"C"へ進む。

【００２４】

図５のブロック１３において、レベル情報が正しいか否かが判断される。正しくなければ、レベル・サイズを計算し(ブロック１５)、ポインタ・フラグを「１」にセットし(ブロック１７)、このレベルのため一時的記憶ブロックを割り当てる(ブロック１９)。次に、メモリが正しく割り当てられたことを確認する(ブロック２１)。メモリ割り当ての際エラーが発生すればこれ以上処理は行わない。メモリが正しく割り当てられれば、記号"D"へ進む。

【００２５】

上記ブロック１３において、レベル情報が正しいと判断される場合、ブロック２７においてポインタ・フラグが「１」にセットされているか否かが検査される。「１」にセットされていれば、このレベルのための一時メモリが解放され(ブロック２８)、ポインタ・フラグを「０」にセットし(ブロック３１)、このレベルに関する連続メモリへのオフセットを計算する(ブロック３３)。次に処理は記号"D"へ進む。ブロック２７において、ポインタ・フラグが「０」にセットされていれば、このレベルに関する基底マップの連続メモリへのオフセットを計算するだけで(ブロック３３)、記号"D"へ進む。

【００２６】

図３のループに戻って、ループが１５回の繰り返しを完了すると、古い連続メモリが解放され(ブロック６０)、処理は記号"D"へ進む。図６の記号"D"へ進んで、レベル情報が記憶され(ブロック３９)、ポインタ・フラグが「１」にセットされているか否かが判断される(ブロック４１)。「１」であれば、処理は終了する(ブロック４７)。ポインタ・フラグが「０」にセットされていれば、オフセットが０を越えているかどうかが更に検査される。オフセットが０であれば、処理は終了する(ブロック４７)。ポインタ・フラグが「０」にセットされていて、オフセットが０を越えていれば、このレベルに関するオフセットを記憶しなければならない(ブロック４５)。次に処理は終了する(ブロック４７)。

【００２７】

以下の表１ないし表４に、本発明の動作様態を示すいくつかの特定の例を示す。以下の諸例の各々において、ダウン・ロード・シーケンスは所与であり、各ダウン・ロード・シーケンスは、そのレベルに関するレベル番号、幅×高さのレベル・サイズおよびポインタ・フラグ値を含む。

【００２８】

【表１】

例１

レベル番号	レベル・サイズ	ポインタ・フラグ
０	８×８	０
１	４×４	０
２	２×２	０
３	１×１	０

【００２９】

例１において、レベル０、１、２および３に対応する４つのレベル・マップがダウンロードされ、それらはそれぞれ８×８、４×４、２×２および１×１マップである。従って、レベル０(すなわち基底マップ)に関する最初のレベル・マップのダウンロードに際して連続的メモリの全ブロックを割り当てることができるので、問題はない。すべてのレベルが順序正しくダウンロードされ、レベル・サイズのすべてが完全に整合しているので、これ

は最も簡単な場合である。

【００３０】

【表２】

例２

レベル番号	レベル・サイズ	ポインタ・フラグ
2	１×１	0
1	２×２	0
0	４×４	0

【００３１】

例２においては、レベル２、１および０に対応する３つのレベル・マップがダウンロードされ、それらはそれぞれ１×１、２×２および４×４マップである。

最初のレベル２に関するレベル・マップがダウンロードされる際、連続的メモリ・ブロック全体を割り当てることが可能であり、また、レベル番号およびレベル・サイズのすべてが整合しているので、問題に出会うことはない。

【００３２】

【表３】

例３

レベル番号	レベル・サイズ	ポインタ・フラグ
0	４×４	0
1	１６×１６	1
2	１×１	0
1	２×２	0

【００３３】

例３においては、レベル０、１、２および１に対応する４つのレベル・マップがダウンロードされ、それらはそれぞれ４×４、１６×１６、１×１および２×２マップである。最初のレベル０に関するレベル・マップがダウンロードされる時、連続的メモリ・ブロック全体が割り当てられるが、その際次に２×２マップのレベル１、４×４マップのレベル２等々のレベル・マップを受け取るであろうと仮定される(注：受け取るマップの数は不明である)。しかしながら、１６×１６のレベル１のダウンロードの際、整合性のないことが検出される。従って、レベル１の１６×１６マップは一時的メモリに記憶され、問題が存在することを示すためポインタ・フラグは１にセットされる。更に第３および第４のレベルがダウンロードされる時、それら(１×１および２×２マップ)は最初のレベル０と整合性があるので、問題は解消する。従って、元のダウンロードされた１６×１６の「レベル１」は破棄し、一時的メモリを解放することができる。注：この例では、最初のレベルのダウンロードに基づく初めの連続メモリ・ブロックが使用されている。

【００３４】

【表４】

例４

レベル番号	レベル・サイズ	ポインタ・フラグ
0	８×８	0
1	２×２	1
2	１×１	1
0	４×４	0

【００３５】

例４においては、レベル０、１、２および０に対応する４つのレベル・マップがダウンロードされ、それらはそれぞれ８×８、２×２、１×１および４×４マップである。レベル０(すなわち基底マップ)に関する最初のレベル・マップがダウンロードされる際、連続メモリ・ブロック全体が割り当てられる。しかしながら、次にダウンロードしされるレベル１のサイズはレベル０と整合していないので、レベル１(２×２マップ)データは一時的メモリに記憶され、ポインタ・フラグが１にセットされる。３番目のレベルであるレベル２

は、１×１マップであり、これもまた最初のレベル０と整合性がないため、一時的メモリに記憶され、ポインタ・フラグが１にセットされる。最後の第４の新たなレベル・ゼロは既にロードされているレベル１およびレベル２と整合性があるので、本発明に従って、それらレベル１およびレベル２は、第４の新しいレベル０と共に、連続メモリへ記憶される。

【００３６】

上述の通り、本発明の方法は、どのレベルのマップの受領に際してもすべてのレベルに対して連続的メモリ・ブロックを割り当てる手段を提供する。追加のレベル・マップがダウンロードされる時、本発明は、先行したメモリ割り当てと整合性があることを確認し、整合性がない場合、整合性のあるレベルのマップがダウンロードされるまで、それら整合性のないマップ情報を一時的メモリに記憶する方法を提供する。整合性のあるレベル・マップ・セットの受領を確認すると、すべての整合性のあるレベル・マップのデータは単一の連続メモリ・ブロックへ記憶される。

【００３７】

以下の表５ないし表１１は、本発明の詳細を更に理解する上での参照のため、本発明の好ましい実施形態に関してＣプログラミング言語で書かれたサンプル・ソース・コードを含む。以下のプログラムの著作権はヒューレット・パッカード・カンパニーにあり、ヒューレット・パッカード・カンパニーの書面による承諾なしに、保存目的の場合を除いて、これらプログラムを複製することは禁止されている。

【００３８】

【表５】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include "mem_alloc.h"
#include "GL/gl.h"
#include "ogl_types.h"
#include "ogl_state.h"
#include "ogl_texture.h"
#include "ogl_env.h"
#include "ds_public.h"
#include "ocm_cmd.h"
#include "pcm_cmd.h"
/*
**********************************************************************
**
** 目的 - 完全なＭＩＰマップのために必要なメモリの計算
** 入力 - w - マップ幅
**        h - マップの高さ
**        myBorder - マップに関するボーダー・サイズ
**        tSize - 内部記憶のためのテクセル・サイズ
**        level - レベル番号
**        mipLevel - ＭＩＰレベルの総数－１を戻す
**        offset - 計算されたオフセットを戻すために使用される
** 戻り値 - 計算されたメモリ・サイズ
**********************************************************************
*/
static int32 computeMipMemSize(
int32 w,
```

```
int32 h,
int32 myBorder,
int32 tSize,
int32 level,
uint16 *mipLevel,
int32 *offset)
{
int bSize;
int memSize;
int numLevels;
int high;
/*
* これは１×１までの完全なＭＩＰマップのサイズを計算する。
*/
memSize = 0;
high = (w 〉 h)? w:h;
for (numLevels = 0; high 〉= 1; high 〉〉= 1)
{
/*
* 現在時レベルのオフセットをメモリに保存する。
*/
if(numLevels == level)
*offset = memSize;
/*
* 現在時レベルのためのメモリを計算する。
*/
bSize = (2 * myBorder) * w + (2 * myBorder) * h +
(myBorder * myBorder) * 4;
memSize += tSize * (w * h + bSize);
```

```
w = (w + 1)/2;
h = (h + 1)/2;
numLevels++;
}
*mipLevel = numLevels;
return(memSize);
}
```

【００３９】
【表６】

```
/*
**********************************************************************
**
** 目的 - マップの現在時レベルが、基底マップの高さ／幅、
**        ボーダー・サイズおよび形式と同じか検査する。
**
** 入力 - target - テクスチャ・マップの次元
**        mipLevel - 基底レベル情報
**        width - 現在時レベルに対応する基底マップの(計算された)幅
**        height - 現在時レベルに対応する基底マップの
**                 (計算された)高さ
**        border - 現在時レベルのボーダー・サイズ
**        internalFormat - 現在時レベル内部形式
** 戻り値 - レベルがＯＫなら１、さもなければ０
**********************************************************************
*/
static int32 checkLevel(
Enum target,
Int32 level,
MipLevelState mipLevel,
Uint32 width,
Uint32 height,
Uint32 border,
Enum internalFormat)
{
int levelOk;
Uint32 tmpW;
Uint32 tmpH;
levelOk = 1;
```

```
tmpW = mipLevel.width >> level;
if(tmpW == 0)
tmpW = 1;
if(tmpW!=width)
level0k = 0;
if(target == GL_TEXTURE_2D)
{
tmpH = mipLevel.height >> level;
if(tmpH == 0)
tmpH = 1;
if(tmpH! = height)
level0k = 0;
}
if(mipLevel.border! = border)
level0k = 0;
if(mipLevel.internalFormat!= internalFormat)
level0k = 0;
return(level0k);
}
```

【００４０】

【表７】

```
/*
**********************************************************************
**
** 目的 - ＭＩＰマップレベルのために必要とされるメモリを計算する。
**
** 入力 - width - 基底レベルの幅
**        height - 基底レベルの高さ
**        tSize - 内部記憶のためのテクセル・サイズ
**        myBorder - 内部記憶のためのボーダー・サイズ
** 戻り値 - 計算されたレベル・サイズ
**********************************************************************
*/
Int32_hpOcm_computeLevelSize(
Int32 width,
Int32 height,
Int32 tSize,
Int32 myBorder)
{
int bSize;
int levelSize;
bSize = (2 * myBorder) * width + (2 myBorder) *
height + (myBorder * myBorder) * 4;
levelSize = tSize * (width * height + bSize);
return(levelSize);
}
```

【００４１】
【表８】

```
/*
**************************************************************************
** 目的 - 現在時レベルに関する基底マップからのテクセル・オフセット
**        を計算する。
**
** 入力 - width - 基底レベルの幅
**        height - 基底レベルの高さ
**        tSize - 内部記憶のためのテクセル・サイズ
**        myBorder - 内部記憶のためのボーダー・サイズ
**        level - ＭＩＰマップのレベル
**
** 戻り値 - 計算されたオフセット
**************************************************************************
*/
Uint32_hp0cm_computeTexelOffset(
Int32 width,
Int32 height,
Int32 tSize,
Int32 myBorder,
Int32 level)
{
int i;
unsigned int offset;
offset = 0;
/*
* 現在時レベルより小さいレベルの各々のサイズを合計する。
*/
for(i=0; i<level; i++)
{
```

```
offset += _hpOcm_computeLevelSize(width, height, tSize, myBorder);
width = (width + 1)/2;
height = (height + 1)/2;
}
return(offset);
}
```

【００４２】

【表９】

```
/*
**********************************************************************
**
** 目的 - テクスチャ・マップのための内部記憶域を割り当てる。
**
** 入力 - memSize - 割り当てるメモリのサイズ
**        *sharedID - 戻り値としての共有メモリＩＤ
**        *ptr - 割り当てられたメモリ・ポインタを記憶するために
**               使用される
**        *sharedTexels - 共用メモリが割り当てられたことを示す
**                        フラグ
**        targetIndex - マップの次元
**
** 戻り値 - 成功なら０
**        - 失敗なら１
** 環境変数 - OGL_TXTR_SHMEM_THRESHOLD
**            この環境変数は、プロセス・メモリ対共用メモリの
**            使用のためのフェンスをセットする。
**            しきい値(threshold)より大きいサイズを持ついかなる
**            テクスチャも共用メモリに記憶される。
**            初期値は、１０２４＊１０２４バイトにセットされる。
**
**アルゴリズム - 要求されたメモリが共用メモリ・フェンスより
**               大きい場合、共用メモリ・プールからメモリを
**               割り当て、さもなければ通常のメモリを割り当てる。
**
**********************************************************************
*/
Int32 allocateTexelMemory(
```

```
Uint32 memSize,
Uint32 *sharedID,
Texel*ptr,
unsigned *sharedTexels,
Enum targetIndex)
{
int shmID;
char *tmpPtr;
char *cp;
long int tmp_shared_memory_limit;
if(memSize == 0)
{
ptr->lum8 = (TexelLum8 *)NULL;
*sharedTexels = 0;
return(0);
}
/*
** 可能な最適化：値がPCM状態である環境変数の値を記憶する。
** それはただ１つのgenenvが存在する場合である。
*/
cp = OglGetenv(HPOGLINT_TXTR_SHMEM_THRESHOLD);
if(cp)
{
tmp_shared_memory_limit = (long int)atol(cp);
/*
** 負数を１へクランプする
*/
if(tmp_shared_memory_limit < 1)
tmp_shared-memory_limit = 1L;
```

```
}
else
tmp_shared_memory_limit = OGL_TXTR_SHMEM_DEFAULT;
shmID = -1;
/*
** memSizeがしきい値より大きいかこれが１Ｄマップでない場合、
** 共用メモリを割り当てる。１Ｄマップは使用しない。
*/
if((memSize >= tmp_shared_memory_limit)&&
(targetIndex!= OGL_TEXTURE_1D))
{
shmID = shmget(IPC_PRIVATE,memSize,IPC_CREAT|0666);
if(shmID!=-1)
{
tmpPtr = (char*)shmat(shmID,(char*)0,0);
if((int)tmpPtr == -1)
{
/*
** 共用メモリＩＤを除去する
*/
shmctl(shmID,IPC_RMID,0);
shmID = -1;
}
else {
/*
** 有効な共用メモリＩＤをメモリに添付する。
** shmIDおよびメモリ・ポインタを保存し、成功を意味する１を戻す。
*/
*sharedID shmID;
```

```
ptr->lum8 = (TexelLum8*)tmpPtr;
*sharedTexels = 1;
return(0); /* 共用メモリ成功 */
}
}
}
if(shmID == -1)
{
/*
** 通常メモリの割り当てを試みる
*/
tmpPtr = (char*)SumMalloc(memSize, FREE_MANUALLY);
if(tmpPtr == (char*)NULL)
{
/*
** メモリなし、テクスチャ・マッピングなし
*/
SET_OGL_ERROR(GL_OUT_OF_MEMORY);
return(-1);
}
else
{
/*
** メモリ割り当て可能。メモリ・ポインタを保存し、
** 通常メモリ成功を示す０を戻す。
*/
ptr->lum8 = (TexelLum8*)tmpPtr;
*sharedTexels = 0;
return(0); /* 通常メモリ成功 */

}
}
return(0);
}
```

【００４３】

【表１０】

```
/*
**********************************************************************
**
** 目的 - 内部メモリ解放
**
** 入力- sharedTexel - 共用メモリを示すフラグ
**       sharedID - 共用メモリのID
**       *ptr - メモリ・ポインタ
**
**********************************************************************
*/
void_hpOcm_freeTexelMemory(
unsigned sharedTexel,
Uint32 sharedID,
void *ptr)
{
if(sharedTexel)
{
shmdt(ptr);
shmctl(sharedID,IPC_RMID,0);
}
else
{
if(ptr! = (void*)NULL)
SumFree(ptr);
}
}
#pragma inline Log2
static int Log2(int Value)
```

```
{
int Result = 0;
while( Value > 1 ){
Result++, Value >>= 1;
}
return Result;
}
```

【表１１】

```
/*
**********************************************************************
**
** 目的 - このルーチンは、テクスチャ・マップのための内部
**        グラフィック・コアメモリを割り当て、ピクセルをアンパック
**        して転送するために２Ｄパイプを実行し、各テクスチャ・
**        マップに関してダウンロードされた有効な
**        mipmap(ＭＩＰマップ)レベルの数を把握し続ける。すべての
**        レベルのＭＩＰマップを連続メモリに保持し続けるため、
**        このルーチンは複雑である。
**
** 入力 - target - GL_TEXTURE_1DまたはGL_TEXTURE_2Dを指定する
**        level - 細部レベル番号
**        internalFormat - ユーザ指定内部形式
**        width - テクスチャの幅 ** ボーダーを含まない **
**        height - テクスチャの高さ ** ボーダーを含まない **
**        border - テクスチャ・マップ・ボーダーのサイズ
**
** 呼び出し元：glTexImage*D
**
** アクセスされる大域変数：_hpOgl_context
**
** アルゴリズム - 基底マップがメモリを持っていなければ、
**                ＭＩＰマップに十分なメモリを割り当てる。
**                到来レベルの形式または境界サイズが、現在時
**                基底レベルのものと合致しない場合は、マップの
**                残りとは別のメモリを割り当て、そのレベルに
**                マークをつける。
**
```

```
** 呼び出される関数 - getInternalFormatAndSize
**                   computeMipMemSize
**                   allocateTexelMemory
**                   checkLevel
**                   DS_NO_LOCK.destroyTexture
**                   _hpOcm_computeLevelSize
**                   _hpOcm_computeTexelOffset
**                   memcpy
**                   SumFree
**                   SumMalloc
**                   _hpOcm_freeTexelMemory
**                   storeMagicBorderValue
**
** エラー・メッセージ - GL_OUT_OF_MEMORY
**
*/
static void downLoadTexels(
Enum target,
Int32 level,
Int32 internalFormat,
Int32 width,
Int32 height,
Int32 border)
{
TextureObjectPtr boundTexels;
MipLevelStatePtr baseMap;
MipLevelStatePtr thisLevel;
Int32 offset;
Int32 tSize;
```

```
Int32 level0k;
Int32 ourType;
Uint32 memSize;
int i;
int myBorder;
unsigned tmpSharedTexels;
Enum targetIndex;
Int32 ignore[4];
/*
** たとえユーザがテクセルについてボーダーを持たないとしても、
** 本方法はボーダーに対して常に空間を割り当てるので
** myBorderを１にセットする。オフセットは０に初期設定する。
*/
myBorder = INTERNAL_BORDER_SIZE;
offset = 0;
/*
** 正しい次元の現在時テクスチャへのポインタを入手する。
** baseMapおよびthisLevelをセットする。
*/
if(target == GL_TEXTURE_2D)
targetIndex = 0GL_TEXTURE_2D;
else
{
targetIndex = 0GL_TEXTURE_1D;
if(width 〉 0)
height = 1;
else
height = 0;
}
```

```
boundTexels = OGL_TEXOBJ.boundTextures[targetIndex];
baseMap = &boundTexels->level[0];
thisLevel = &boundTexels->level[level];
/*
** 現在時テクスチャが表示リスト最適化の一部であれば、
** 表示リスト・ポインタを変更しないようにテクセル情報を
** 新しいポインタへコピーする
*/
if(boundTexels->hdr.dlmFlag == 1)
Texel texelSave;
texelSave.lum8 = baseMap->texelData.lum8;
memSize = computeMipMemSize(baseMap->width,
baseMap->height,
myBorder,
baseMap->bytesPerTexel,
0,
&boundTexels->hdr.numMipLevels,
&offset);
if(-1 = allocateTexelMemory(memSize,
&boundTexels->hdr.sharedTexelIID,
&baseMap->texelData,
&tmpSharedTexels,
boundTexels->targetIndex))
{
/*
** エラーはすでにセットされている。
*/
return;
}
```

```
boundTexels->hdr.sharedTexels = tmpSharedTexels;
baseMap->pointerFlag =!TMP_MEMORY_POINTER;
memcpy(baseMap->texelData.lum8, texelSave.lum8, memSize);
boundTexels->hdr.dlmFlag = 0;
}
DS_NO_LOCK.getInternalFormatAndSize(internalFormat, &ourType,
&tSize, ignore);
/*
** 既存のテクセル・マップ・メモリを検査する。
** 図２ないし図６の流れ図はここから始まる。
*/
if(baseMap->texelData.lum8 == (TexelLum8 *)NULL)
{
Uint32 wSave;
Uint32 hSave;
/*
** 基底レベルのサイズを計算する。この情報を使用して、
** 現在時レベルが正しいＭＩＰレベルであるか、
** メモリ割り当てを行うべきものかを確認する。
*/
wSave = width;
hSave = height;
if(level!= 0)
{
wSave <<= level;
if(target == GL_TEXTURE_2D)
hSave <<= level;
}
/*
```

```
** テクセル・データに必要な全メモリを計算する。このコードは、
** (サイズ１の)ボーダーを含む完全ＭＩＰマップに対して十分な
** メモリを割り当てる。
*/
memSize = computeMipMemSize(wSave, hSave, myBorder, tSize,
level,
&boundTexels->hdr.numMipLevels, &offset);
if(-1 == allocateTexelMemory(memSize,
&boundTexels->hdr.sharedTexelID,
&baseMap->texelData,
&tmpSharedTexels,
targetIndex))
{
/*
** エラーは既にセットされている。
*/
return;
}
boundTexels->hdr.sharedTexels = tmpSharedTexels;
baseMap->texelType = ourType;
baseMap->internalFormat = internalFormat;
baseMap->bytesPerTexel = tSize;
baseMap->border = border;
baseMap->width = wSave;
baseMap->height = hSave;
baseMap->widthLog2 = Log2(wSave);
baseMap->heightLog2 = Log2(hSave);
baseMap->pointerFlag =!TMP_MEMORY_POINTER;
baseMap->depth = 0;
```

```
baseMap->depthLog2 = 0;
}
/*
** メモリは今割り当てられたか、あるいは既に存在していた。
** レベルにテクセル情報を含める。このレベルが有効なＭＩＰレベルで
** あるか検査する。この検査には、同じ境界サイズ持っているか、
** 基底レベルと同じ内部形式か、正しい高さおよび幅の値を
** 持っているかが含まれる。
*/
levelOk = checkLevel(target,level, *basemap, width, height,
border,internalFormat);
/*
** テクスチャが結合されていれば、ＤＳＭ(ハードウェア)は、
** これらのテクセルが変更されつつあることを知っている。
*/
if((boundTexels->hdr.texelID!= INVALID_TEXE_ID) && (!levelOk))
DS_NO_LOCK.destroyTexture(boundTexels);
if(level == 0)
{
if(!levelOk)
{
/*
** テクセルを記憶するためのメモリは割り当てられたが、基底マップの
** サイズが変更された。従って、古いメモリ・ポインタを保存し、
** 新しいメモリを割り当て、有効な古いマップのレベルを新しいマップ
** にコピーし、残りのレベルをtmpポインタにコピーし、
** 次に古いメモリを解放する。
*/
Texel texelSave;
```

```
unsigned sharedTexelsSave;
Uint32 sharedTexelIDSave;
/*
** 基底レベルへ新しい値を入れる
*/
baseMap->texelType = ourType;
baseMap->internalFormat = internalFormat;
baseMap->bytesPerTexel = tSize;
baseMap->border = border;
baseMap->width = width;
basemap->height = height;
baseMap->widthLog2 = Log2(width);
baseMap->heightLog2 = Log2(height);
baseMap->pointerFlag = !TMP_MEMORY_POINTER;
baseMap->depth = 0;
baseMap->depthLog2 = 0;
/*
** メモリを解放する前に、現在時テクセル・ポインタ、共用メモリＩＤ
** および共用メモリ・フラグを保存する。
*/
texelSave.lum8 = baseMap->texelData.lum8;
sharedTexelsSave = boundTexels->hdr.sharedTexels;
sharedTexelIDSave = boundTexels->hdr.sharedTexelID;
memSize = computeMipMemSize(width,
height,
myBorder, tSize,
level,
&boundTexels->hdr.numMipLevels,
&offset);
```

```
if(-1 == allocateTexelMemory(memSize,
&boundTexels->hdr.sharedTexelID,
&baseMap->texelData,
&tmpSharedTexels, targetIndex))
{
/*
** エラーは既にセットされている
*/
return;
}
boundTexels->hdr.sharedTexels = tmpSharedTexels;
/*
** 既存のマップにおけるレベルの各々毎に、新しいマップへ
** コピーできるレベルを決定する。残りのレベルはtmpポインタへ
** コピーしポインタ・フラグをセットする。
*/
for(i=1; i<OGL_MAX_MIPMAP_LEVELS; i++)
{
int leveSize;
MipLevelStatePtr curLevel;
curLevel = &boundTexels->level[i];
if(curLevel->texelData.lum8!= NULL)
{
levelSize = _hpOcm_computeLevelSize(curLevel->width,
curLevel->height,
curLevel->bytesPerTexel,
myBorder);
levelOk = checkLevel(target,
i,
```

```
*baseMap,
curLevel->width,
curLevel->height,
curLevel->border,
curLevel->i nternalFormat)
if(level0k)
{
offset = _hpOcm_computeTexelOffset(baseMap->width,
baseMap->height,
baseMap->bytesPerTexel,
myBorder, level);
memcpy(baseMap->texelData.lum8 + offset,
curLevel->texelData.lum8,
levelSize);
if(curLevel->pointerFlag == TMP_MEMORY_POINTER)
{
SumFree(curLevel->texelData.lum8);
curLevel->pointerFlag = !TMP-MEMORY-POINTER;
curLevel->texelData.lum8 = baseMap->texelData.lum8 +
offset;
}
else
{
Texel levelSave;
/*
** このレベルについてポインタ・フラグが既にセットされていれば、
** このレベルをコピーする理由はない。
*/
if(curLevel->pointerFlag!= TMP_MEMORY_POINTER)
```

```
{
level1Save.lum8 = curLevel->texelData.lum8;
curLevel->pointerFlag = TMP_MEMORY_POINTER;
curLevel->texelData.lum8 =
(TexelLum8 *)SumMalloc(level1Size,
FREE_MANUALLY);
if(curLevel->texelData.lum8 == (TexelLum8 *)NULL)
{
SET_OGL_ERROR(GL_OUT_OF_MEMORY);
return;
}
memcpy(curLevel->texelData.lum8,
level1Save.lum8,
level1size);
}
}
}
}
/*
** texelSaveポインタにおける古いメモリを解放する。
*/
_hpOcm_freeTexelMemory(sharedTexelsSave,
sharedTexelIDSave,
(void *)texelSave.lum8);
}
}
else
{
/*
```

```
** 当該レベルは基底レベルでない。
*/
if(level0k)
{
if (thisLevel->pointerFlag == TMP_MEMORY_POINTER)
{
/*
** 以前ＯＫではなかったこのレベルが今やＯＫである。
** 従って、tmpメモリを解放し、pointerFlagをリセットし、
** 基底マップによってポイントされているＭＩＰアレイへの
** オフセットを再計算する。
*/
SumFree(thisLevel->texelData.lum8);
thisLevel->pointerFlag = !TMP_MEMORY_POINTER;
offset = _hpOcm_computeTexelOffset(baseMap->width,
baseMap->height,
baseMap->bytesPerTexel,
myBorder, level);
}
else
{
/*
** このレベルは既に初期化されていることは絶対にないので、
** 基底マップによってポイントされているＭＩＰアレイへの
** オフセットを計算する。
*/
offset = _hpOcm_computeTexelOffset(baseMap->width,
baseMap->height,
baseMap->bytesPerTexel,
```

```
myBorder, level);
}
}
else
{
/*
** レベルはＯＫでない。従って、一時メモリを割り当て、そこへ
** このレベルのテクセルを記憶する。また、レベルが基底マップへの
** オフセットではないことを示すpointerFlagをセットする。
*/
int levelSize;
levelSize = _hpOcm_computeLevelSize(width, height, tSize,
myBorder);
thisLevel->pointerFlag = TMP_MEMORY_POINTER;
thisLevel->texelData.lum8 = (TexelLum8 *)SumMalloc(levelSize,
FREE_MANUALLY);
if(thisLevel->texelData.lum8 == (TexeLum8*)NULL)
{
SET_OGL_ERROR(GL_OUT_OF_MEMORY);
return;
}
}
}
/*
** レベル情報を保存する。
*/
thisLevel->texelType = ourType;
thisLevel->internalFormat = internalFormat;
thisLevel->bytesPerTexel = tSize;
```

```
thisLevel->border = border;
thisLevel->width = width;
thisLevel->height = height;
thisLevel->widthLog2 = Log2(width);
thisLevel->heightLog2 Log2(height);
thisLevel->depth = 0;
thisLevel->depthLog2 = 0;
/*
** このレベルがＯＫであれば、このレベルに関するオフセットを
** texelDataに記憶する。注：これがレベル０であればオフセットは
** 常にゼロである。
*/
if((thisLevel->pointerFlag! = TMP_MEMORY_POINTER) && offset)
thisLevel->texelData.lum8 = baseMap->texelData.lum8 + offset;
if(thisLevel->texelData.lum8)
storeMagicBorderValue(tSize,(char*)thisLevel->texelData.lum8);
}
```

【００４４】

本発明には、例として次のような実施様態が含まれる。

（１）コンピュータ・グラフィックス・システムにおいて連続的メモリ・ブロックをテクセル・データへ割り当てる方法であって、ダウンロードされる最初のレベル・マップに関するテクセル・データに対応するデータを受け取るステップと、上記ダウンロードされる最初のレベル・マップのサイズおよびレベル番号に基づいて、完全なＭＩＰマップを記憶することのできる連続的メモリ・ブロックを割り当てるステップと、上記ダウンロードされる最初のレベル・マップに関する上記テクセル・データのレベル番号に基づいて、上記ダウンロードされる最初のレベル・マップに関連するデータを記憶する上記連続的メモリ・ブロックのあらかじめ定められた適切な位置を表すオフセット値を決定するステップと、上記ダウンロードされる最初のレベル・マップに関する上記テクセル・データを上記オフセット値によってポイントされる連続的メモリ・ブロック内の位置に記憶するステップと、上記ダウンロードされる最初のレベル・データを受け取った後、更に別のレベル・マップに関する付加的テクセル・データを繰り返し受け取り、それら受け取ったレベル・データのそれぞれ毎に、当該レベル・データが上記ダウンロードされた最初のテクセル・データと整合しているか否かを確認し、整合している場合上記連続的メモリ・ブロックに対するオフセットを計算してそのオフセットによってポイントされる上記連続的メモリ・ブロック内の位置へ当該テクセル・データを記憶し、整合していない場合、当該テクセル・データを一時的メモリへ記憶すると共にテクセル・データが一時的メモリにあることを示す値にポインタ・フラグをセットし、すべての上記付加的レベル・データについて上記処理を繰り返した後、上記一時メモリを連続的メモリ・ブロックへ移動することによって整合性のあるテクセル・データのセットが完成する場合にはそれらテクセル・データを連続的メモリ・ブロックへ移動するステップと、を含むテクセル・データへの連続的メモリ割り当て方法。

【００４５】

（２）コンピュータ・グラフィックス・システムにおける連続的メモリにテクセル・データをダウンロードする方法であって、ダウンロードされる最初のテクセル・データのＭＩＰマップのレベル番号およびサイズを基にして、当該テクスチャに関連するすべてのＭＩＰマップに必要な連続的メモリ量を割り当てるステップと、上記最初のＭＩＰマップに関するデータを、レベル番号およびサイズを基にして計算するオフセット値に従ってそのレベル番号にとって適切な上記連続的メモリ位置に記憶するステップと、更にいくつかの後続のＭＩＰマップがダウンロードされるにつれて、それらの各々が上記最初にダウンロードされたＭＩＰマップとレベル番号およびサイズの観点から整合しているか否かを判断するステップと、上記判断の結果整合していれば、それらレベル番号に基づいて計算されるオフセット値が示す上記連続的メモリ位置へそれらレベルのデータを記憶するステップと、上記判断の結果整合していない場合、それらレベルのデータを一時的メモリへ記憶するステップと、上記後続のＭＩＰマップにおいて基底レベルのデータがダウンロードされる毎に、上記一時的メモリに記憶されたデータを検査して、当該基底レベルと整合性があるレベルのデータがあれば、それらを当該基底レベルが記憶される連続的メモリへ移動するステップと、を含む、連続的メモリへのテクセル・データのダウンロード方法。

【００４６】

【発明の効果】

本発明によって、データの完全性を維持しながら、ＭＩＰ化されたテクスチャ・マップの複数レベルを連続的メモリに記憶することが可能となり、それによって、完全にＭＩＰ化されたテクスチャ・マップをハードウェアにダウンロードする際あるいはテクスチャ・マッピングがソフトウェア・ラスタライゼーションを使用する際のメモリ・キャッシュ・ミスが減少あるいは除去される。

【図面の簡単な説明】

【図１】テクスチャＭＩＰマップのセットを示す図式である。

【図２】図３、図４、図５および図６と共に、本発明の方法の動作を示す流れ図である。

【図３】図２、図４、図５および図６と共に、本発明の方法の動作を示す流れ図である。

【図４】図２、図３、図５および図６と共に、本発明の方法の動作を示す流れ図である。

【図５】図２、図３、図４および図６と共に、本発明の方法の動作を示す流れ図である。

【図６】図２、図３、図４および図５と共に、本発明の方法の動作を示す流れ図である。

【符号の説明】

１００　基底マップ(レベル０マップ)

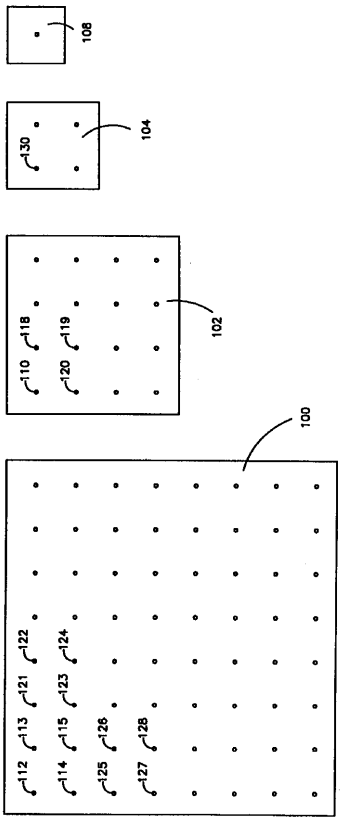
１０２　レベル１マップ

１０４　レベル２マップ

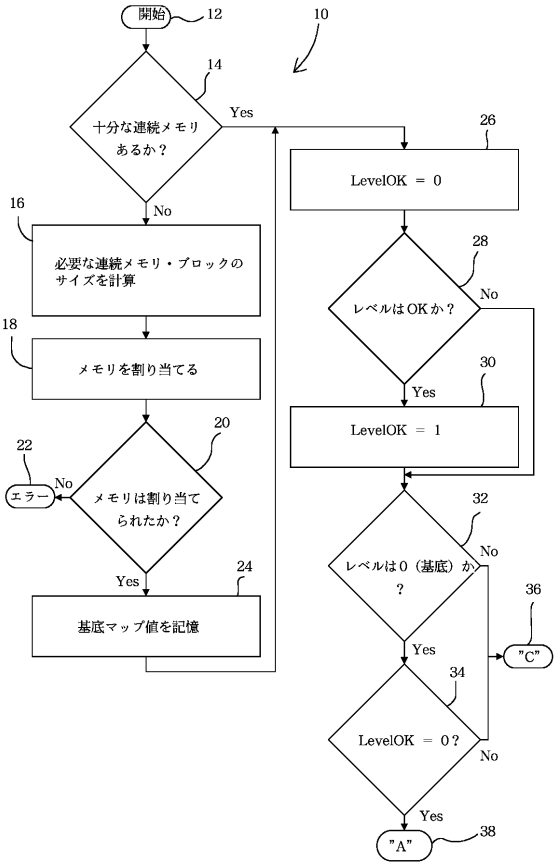
１０８　レベル３マップ

１１０、１１２、１２０、１２８　テクセル

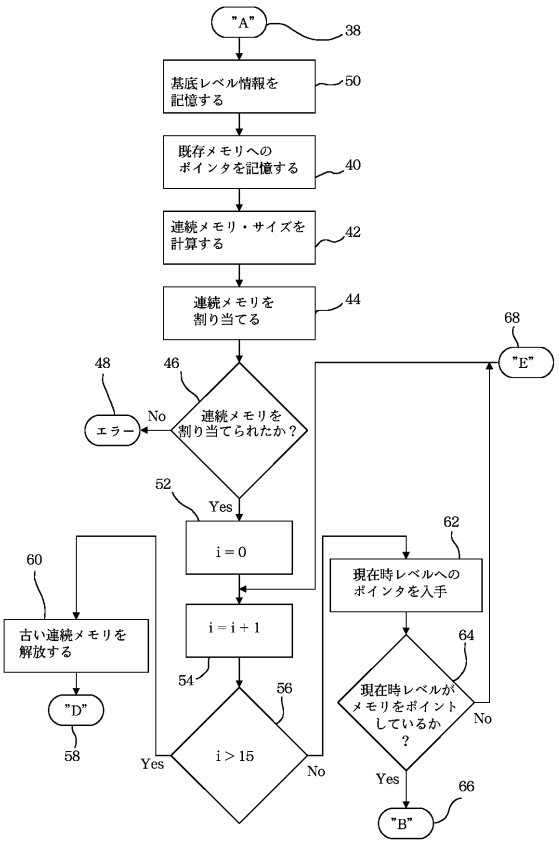
【図１】


108
104
130
102
118
119
110
120
100
122
124
121
123
113
115
126
128
112
114
125
127

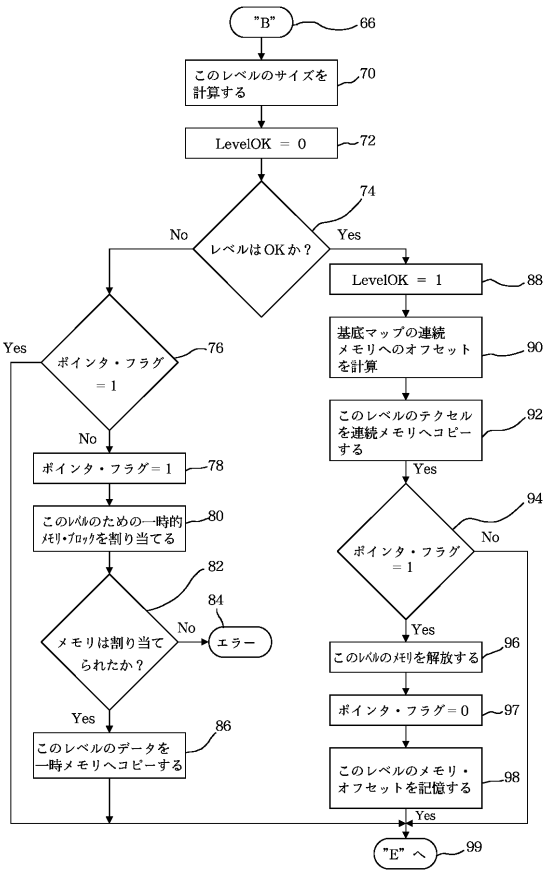
【図２】


開始
12
10
14
十分な連続メモリあるか？
Yes
No
26
LevelOK = 0
16
必要な連続メモリ・ブロックのサイズを計算
28
レベルはOKか？
No
Yes
18
メモリを割り当てる
30
LevelOK = 1
22
エラー
No
20
メモリは割り当てられたか？
32
レベルは0（基底）か？
No
Yes
24
基底マップ値を記憶
36
"C"
Yes
34
LevelOK = 0？
No
Yes
"A"
38

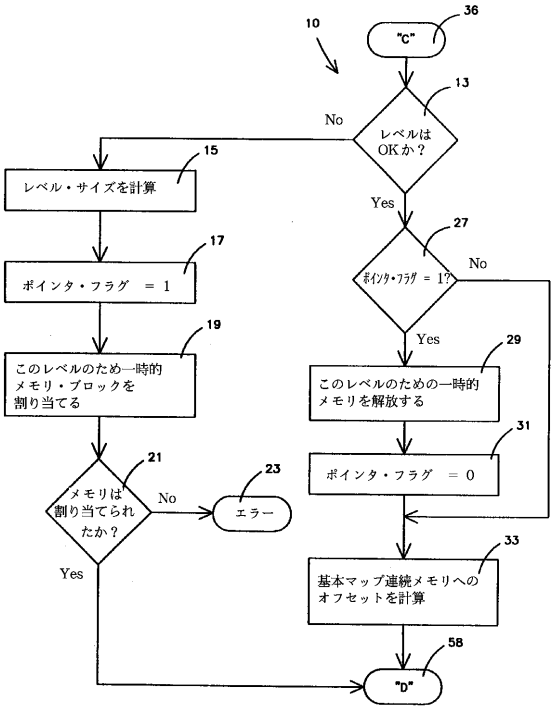
【図３】


"A"
38
基底レベル情報を記憶する
50
既存メモリへのポインタを記憶する
40
連続メモリ・サイズを計算する
42
連続メモリを割り当てる
44
68
"E"
48
46
連続メモリを割り当てられたか？
エラー
No
52
Yes
i = 0
62
現在時レベルへのポインタを入手
60
i = i + 1
古い連続メモリを解放する
64
54
56
現在時レベルがメモリをポイントしているか？
No
"D"
58
i > 15
Yes
No
Yes
66
"B"

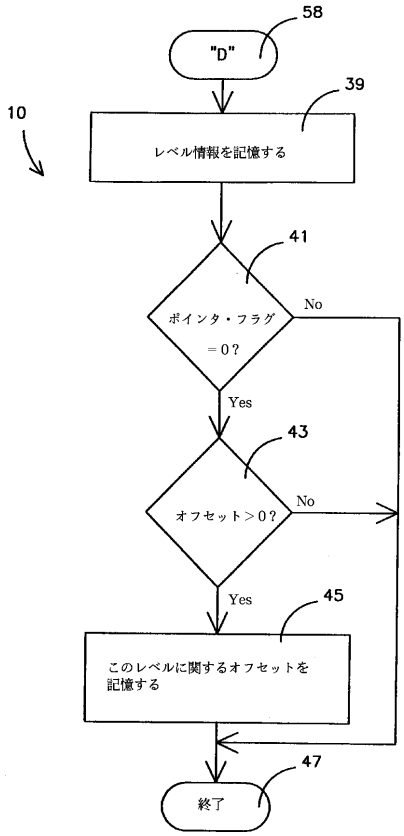
【図４】


"B"
66
このレベルのサイズを計算する
70
LevelOK = 0
72
74
レベルはOKか？
No
Yes
LevelOK = 1
88
Yes
ポインタ・フラグ = 1
76
基底マップの連続メモリへのオフセットを計算
90
No
ポインタ・フラグ = 1
78
このレベルのテクセルを連続メモリへコピーする
92
このレベルのための一時的メモリ・ブロックを割り当てる
80
Yes
94
ポインタ・フラグ = 1
No
82
84
メモリは割り当てられたか？
No
エラー
Yes
このレベルのメモリを解放する
96
ポインタ・フラグ = 0
97
Yes
86
このレベルのデータを一時メモリへコピーする
このレベルのメモリ・オフセットを記憶する
98
Yes
"E" へ
99

【図５】


10
"C"
36
13
レベルは
OKか？
No
Yes
15
レベル・サイズを計算
17
ポインタ・フラグ　＝ 1
19
このレベルのため一時的
メモリ・ブロックを
割り当てる
21
メモリは
割り当てられ
たか？
No
23
エラー
Yes
27
ﾎﾟｲﾝﾀ･ﾌﾗｸﾞ = 1?
No
Yes
29
このレベルのための一時的
メモリを解放する
31
ポインタ・フラグ　＝ 0
33
基本マップ連続メモリへの
オフセットを計算
58
"D"

【図６】


58
"D"
10
39
レベル情報を記憶する
41
ポインタ・フラグ
＝0？
No
Yes
43
オフセット＞0？
No
Yes
45
このレベルに関するオフセットを
記憶する
47
終了

フロントページの続き

(56)参考文献 特開平６－３０９４７２（ＪＰ，Ａ）
特開平４－２１１８７８（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06T11/00-17/50