(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 196 20 847 B4** 2004.10.28

(12) **Patentschrift**

(21) Aktenzeichen: **196 20 847.5**
(22) Anmeldetag: **23.05.1996**
(43) Offenlegungstag: **12.12.1996**
(45) Veröffentlichungstag
der Patenterteilung: **28.10.2004**

(51) Int Cl.[7]: **G06T 15/00**
G06T 1/60

Innerhalb von 3 Monaten nach Veröffentlichung der Erteilung kann Einspruch erhoben werden.

(30) Unionspriorität:
470817 06.06.1995 US

(71) Patentinhaber:
Hewlett-Packard Co. (n.d.Ges.d.Staates Delaware), Palo Alto, Calif., US

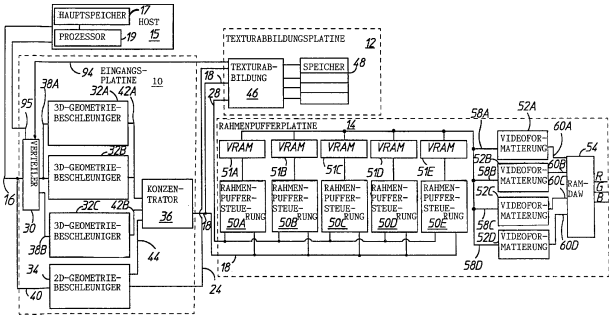
(74) Vertreter:
Schoppe, F., Dipl.-Ing.Univ., Pat.-Anw., 82049 Pullach

(72) Erfinder:
Emmot, Darel N., Fort Collins, Col., US

(56) Für die Beurteilung der Patentfähigkeit in Betracht gezogene Druckschriften:
WO 95 24 682 A1
MESSMER, H.-P.: PC-Hardwarebuch, Addison-Wesley, 1993, S. 233-240;

(54) Bezeichnung: **Verfahren und Vorrichtung zur Texturabbildung**

(57) Hauptanspruch: Verfahren zum Zugreifen auf Texturdaten (S, T) in einem Texturabbildungs-Computergrafiksystem, wobei das System einen Rostcomputer (15) mit einem Hauptspeicher (17) aufweist, der Texturdaten speichert, die eine Mehrzahl von Texeln einschließen, wobei das Verfahren folgende Schritte aufweist:
Speichern zumindest eines Teils der Texturdaten in einem Lokalspeicher (48) des Systems;
temporäres Speichern einer begrenzten Anzahl von Texeln, auf die zuletzt aus dem Lokalspeicher (48) zugegriffen wurde, in einem Texeldatenpuffer (214);
Lesen von Texeln von vorbestimmten Orten (214A0-214D1) des Texeldatenpuffers; und
Zugreifen auf Texel aus dem Lokalspeicher (48), nur wenn solche Texel nicht verfügbar sind, um aus dem Texeldatenpuffer erneut gelesen zu werden.



Beschreibung

[0001] Die vorliegende Erfindung bezieht sich allgemein auf Computergrafiksysteme zur Texturabbildung und insbesondere auf ein Texturabbildungs-Speichersystem und ein Verfahren zum Speichern von Texturabbildungsdaten.

[0002] Computergrafiksysteme werden üblicherweise verwendet, um grafische Darstellungen von Objekten auf einem zweidimensionalen Anzeigeschirm anzuzeigen. Gegenwärtige Computergrafiksysteme können hochdetaillierte Darstellungen liefern und werden in einer Vielzahl von Anwendungen verwendet.

[0003] Bei typischen Computergrafiksystemen wird ein Objekt, das auf dem Anzeigebildschirm dargestellt werden soll, in eine Mehrzahl von Grafik-Grundelementen zerlegt. Grundelemente sind elementare Komponenten eines Grafikbildes und können Punkte, Linien, Vektoren und Vielecke, beispielsweise Dreiecke, einschließen. Typischerweise ist ein Hardware/Software-Schema implementiert, um die Grafik-Grundelemente, die die Ansicht eines oder mehrerer Objekte, die auf dem Bildschirm dargestellt werden, darstellen, auf dem zweidimensionalen Anzeigebildschirm aufzubereiten, oder zu zeichnen.

[0004] Typischerweise werden die Grundelemente, die das dreidimensionale Objekt, das aufbereitet werden soll, definieren, von einem Hostcomputer geliefert, der jedes Grundelement in Form von Grundelementdaten definiert. Wenn das Grundelement beispielsweise ein Dreieck ist, kann der Hostcomputer das Grundelement in der Form der x-, y-, z-Koordinaten seiner Scheitelpunkte ebenso wie der R-, G-, B-Farbwerte jedes Scheitelpunkts definieren. Eine Aufbereitungshardware interpoliert die Grundelementdaten, um die Anzeigebildschirmpixel zu berechnen, die eingeschaltet werden, um jedes Grundelement und die R-, G-, B-Werte für jedes Pixel darzustellen.

[0005] Frühe Grafiksysteme waren nicht in der Lage, Bilder auf eine ausreichend realistische Art und Weise anzuzeigen, um komplexe dreidimensionale Objekte darzustellen oder zu modellieren. Die Bilder, die durch derartige Systeme angezeigt werden, zeigten extrem glatte Oberflächen ohne Texturen, Hügel, Kratzer, Schatten und andere Oberflächendetails, die bei dem Objekt, das modelliert wird, vorliegen.

[0006] Folglich wurden Verfahren entwickelt, um Bilder mit verbesserten Oberflächendetails anzuzeigen. Die Texturabbildung ist ein solches Verfahren, das das Abbilden eines Quellenbilds, das als eine Textur bezeichnet wird, auf einer Oberfläche eines dreidimensionalen Objekts und nachfolgend das Abbilden des mit einer Texturabbildung versehenen, dreidimensionalen Objekts auf dem zweidimensionalen Grafikanzeigebildschirm, um das resultierende Bild anzuzeigen, einschließt. Oberflächendetail-Merkmale, die gewöhnlich mittels einer Textur abgebildet werden, schließen die Farbe, die Spiegelreflexion, die Vektorperturbation, das Spiegelvermögen (specularity), die Lichtdurchlässigkeit, Schatten, Oberflächenunregelmäßigkeiten und die Körnung ein.

[0007] Die Texturabbildung umfaßt das Anwenden eines oder mehrerer Punktelemente (Texel) einer Textur auf jedes Punktelement (Pixel) des angezeigten Abschnitts des Objekts, auf das die Textur abgebildet wird. Die Texturabbildungshardware ist üblicherweise mit Informationen versehen, die die Art und Weise anzeigen, auf die die Texel in einer Texturtabelle den Pixeln auf dem Anzeigebildschirm, der das Objekt darstellt, entsprechen. Jedes Texel in einer Texturtabelle ist durch S- und T-Koordinaten definiert, die den Standort desselben in der zweidimensionalen Texturtabelle identifizieren. Für jedes Pixel wird auf das entsprechende Texel oder die Texel, die auf dasselbe abbilden, aus der Texturtabelle zugegriffen und werden in die endgültigen R-, G-, B-Werte, die für das Pixel erzeugt werden, eingearbeitet, um das mit einer Textur versehene Objekt auf dem Anzeigebildschirm darzustellen.

[0008] Es sollte offensichtlich sein, daß nicht jedes Pixel in einem Objektgrundelement für jede Ansicht des Objekts in einer Eins-Zu-Eins-Entsprechung mit einem einzelnen Texel in der Texturtabelle abbilden muß. Je näher sich das Objekt an dem Sichtfenster, das auf dem Anzeigebildschirm dargestellt wird, befindet, desto größer wird das Objekt beispielsweise erscheinen. Wenn das Objekt auf dem Anzeigebildschirm größer erscheint, wird die Darstellung der Textur detaillierter. Wenn das Objekt einen ziemlich großen Teil des Anzeigebildschirms einnimmt, wird folglich eine große Anzahl von Pixeln verwendet, um das Objekt auf dem Anzeigebildschirm darzustellen, wobei jedes Pixel, das das Objekt darstellt, in einer Eins-Zu-Eins-Entsprechung mit einem einzelnen Texel in der Texturtabelle abbilden kann, oder ein einzelnes Texel auf eine Mehrzahl von Pixeln abbilden kann. Wenn das Objekt jedoch einen relativ kleinen Abschnitt des Anzeigebildschirms verwendet, wird eine viel kleinere Anzahl von Pixeln verwendet, um das Objekt darzustellen, was bewirkt, daß die Textur weniger detailliert dargestellt wird, so daß jedes Pixel auf mehrere Texel abbilden kann. Jedes Pixel kann auch auf mehrere Texel abbilden, wenn eine Textur auf einen kleinen Abschnitt eines Objekts abgebildet wird. Resultierende Texeldaten werden für jedes Pixel berechnet, das auf mehr als ein Texel abbildet, und stellen typischerweise einen Durchschnitt der Texel dar, die auf dieses Pixel abbilden.

[0009] Texturabbildungs-Hardwaresysteme weisen typischerweise einen Lokalspeicher auf, der Daten

speichert, die eine Textur darstellen, die dem Objekt, das aufbereitet wird, zugeordnet ist. Wie oben erläutert wurde, kann ein Pixel auf eine Mehrzahl von Texeln abbilden. Wenn es notwendig wäre, daß die Texturabbildungshardware eine große Anzahl von Texeln, die auf ein Pixel abbilden, von dem Lokalspeicher liest, um einen Durchschnittswert zu erzeugen, wäre eine große Anzahl von Speicherlesevorgängen und die Mittelung der vielen Texelwerte erforderlich, was zeitverbrauchend wäre und das Systemverhalten verschlechtern würde.

[0010] Um dieses Problem zu lösen, wurde ein Schema entwickelt, das die Erzeugung einer Reihe von MIP-Tabellen für jede Textur einschließt, und das Speichern der MIP-Tabellen der Textur, die dem Objekt zugeordnet ist, welches in dem Lokalspeicher der Texturabbildungshardware aufbereitet wird. Eine MIP-Tabelle für eine Textur weist eine Basistabelle auf, die direkt der Texturtabelle entspricht, sowie eine Reihe von gefilterten Tabellen, in denen jede aufeinanderfolgende Tabelle größenmäßig um einen Faktor von zwei in jeder der zwei Dimensionen der Texturtabelle reduziert ist. Ein veranschaulichendes Beispiel eines Satzes von MIP-Tabellen ist in **Fig.** 1 gezeigt. Die MIP-Tabellen (MIP = multum in parvo – viele Dinge an einem kleinen Ort) weisen eine Basistabelle **100** auf, die eine Größe von acht mal acht Texel aufweist, ebenso wie eine Reihe von Tabellen **102**, **104** und **108**, die vier mal vier Texel, zwei mal zwei Texel und ein Texel groß sind.

[0011] Die Vier-Mal-Vier-Tabelle **102** wird durch ein Block-Filtern (Dezimieren) der Basistabelle **100** erzeugt, derart, daß jedes Texel in der Tabelle **102** einem Durchschnitt von vier Texeln in der Basistabelle **100** entspricht. Beispielsweise ist das Texel **110** in der Tabelle **102** gleich dem Durchschnitt der Texel **112** bis **115** in der Tabelle **100**, während die Texel **118** und **120** in der Tabelle **102** jeweils gleich dem Durchschnitt der Texel **121** bis **124** und **125** bis **128** in der Tabelle **100** sind. Die Zwei-Mal-Zwei-Tabelle **104** wird in gleicher Weise durch das Blockfiltern der Tabelle **102** erzeugt, derart, daß das Texel **130** in der Tabelle **104** gleich dem Durchschnitt der Texel **110**, und **118** bis **120** in Tabelle **102** ist. Das einzelne Texel in der Tabelle **108** wird durch das Bilden des Durchschnitts der vier Texel in Tabelle **104** erzeugt.

[0012] Herkömmliche Grafiksysteme laden im allgemeinen die vollständige Reihe von MIP-Tabellen für jede Textur, die mit den Grundelementen, die auf dem Anzeigebildschirm aufbereitet werden sollen, verwendet werden soll, von dem Hauptspeicher des Hostcomputers in den Lokalspeicher der Texturabbildungshardware herunter. Folglich kann die Texturabbildungshardware auf Texturdaten von jeder der Reihe von MIP-Tabellen zugreifen. Die Bestimmung dessen, auf welche Tabelle zugegriffen werden soll, um die Texeldaten für irgendein spezielles Pixel zu liefern, basiert auf der Anzahl von Texeln, auf die das Pixel abbildet. Wenn das Pixel beispielsweise in einer Eins-Zu-Eins-Entsprechung auf ein einzelnes Texel in der Texturtabelle abbildet, wird auf die Basistabelle **100** zugegriffen. Wenn das Pixel jedoch auf vier, sechzehn oder vierundsechzig Texel abbildet, wird auf die Tabellen **102**, **104** bzw. **108** zugegriffen, da diese Tabellen jeweils Texeldaten speichern, die einen Durchschnitt von vier, sechzehn und vierundsechzig Texeln in der Texturtabelle speichern.

[0013] Ein Pixel muß nicht direkt auf ein Texel in der ausgewählten Tabelle abbilden, sondern kann zwischen zwei oder mehr Texel fallen. Einige Grafiksysteme verwenden eine bilineare Interpolation, um Texeldaten exakt zu erzeugen, wenn dies der Fall ist. Wenn ein Pixel zwischen zwei oder mehr Texeleinträge in eine MIP-Tabelle abbildet, sind die resultierenden Texeldaten, die verwendet werden, ein gewichteter Durchschnitt der nächstliegenden Texeleinträge. Folglich können die Texeldaten, die einem beliebigen Pixel entsprechen, der gewichtete Durchschnitt von vier Texeleinträgen in einer einzelnen Tabelle sein. Wenn ein Pixel beispielsweise auf einen Ort in der Tabelle **102** abbildet, der bei **132** angezeigt ist, wären die resultierenden Texeldaten, die diesem Pixel zugeordnet sind, der gewichtete Durchschnitt der Texel **110** und **118** bis **120**.

[0014] Pixel müssen ferner nicht direkt in eine beliebige der Tabellen in der Reihe von MIP-Tabellen abbilden und können zwischen zwei Tabellen fallen. Beispielsweise kann ein Pixel auf eine Anzahl von Texeln in der Texturtabelle abbilden, die größer als Eins jedoch kleiner als Vier ist. Einige Grafiksysteme begegnen dieser Situation durch das Interpolieren zwischen den zwei nächstliegenden MIP-Tabellen, um die resultierenden Texeldaten zu erhalten. Bei dem obigen Beispiel, bei dem ein Pixel auf mehr als ein jedoch weniger als vier Texel in der Texturtabelle abbildet, würden die Texeldaten, die durch die Tabellen **100** und **102** geliefert werden, interpoliert werden, um die resultierenden Texeldaten für das Pixel zu erhalten. Wenn dies mit der oben beschriebenen Interpolation von mehreren Texeleinträgen in einer einzelnen Tabelle kombiniert wird, ist dieses Schema als eine trilineare Interpolation bekannt und kann zu resultierenden Texeldaten für jedes Pixel führen, das als ein gewichteter Durchschnitt von acht Texeln erzeugt wird, d.h. den vier nächstliegenden Texeln in jeder der zwei nächstliegenden Tabellen.

[0015] Wie oben erläutert wurde, erfordern herkömmliche Texturabbildungssysteme das Herunterladen der gesamten Reihe von MIP-Tabellen für jede Textur, die Grundelementen, die durch das System aufbereitet werden sollen, zugeordnet ist, selbst wenn auf einige der MIP-Tabellen nicht zugegriffen wird. Das Herunterladen der MIP-Tabellen, auf die nicht zugegriffen wird, ebenso wie der Abschnitte von

Tabellen, auf die zugegriffen wird, die nicht verwendet werden, ist eine Verschwendung der Betriebsmittel des Systems und reduziert die Bandbreite desselben.

[0016] Außerdem arbeiten einige Texturabbildungssysteme pipelineartig, so daß verschiedene Operationen gleichzeitig auf unterschiedlichen Objektgrundelementen durchgeführt werden. Jedoch kann eine Reihe von MIP-Tabellen für eine Textur groß sein. Die meisten Systeme verwenden einen Lokalspeicher, der in der Lage ist, nur eine derart große Reihe von MIP-Tabellen einzeln zu speichern. Wenn folglich ein Umschalten der Textur, die beim Aufbereiten von Grundelementen verwendet ist, stattfindet, muß das System eine neue Reihe von MIP-Tabellen herunterladen. Typischerweise verläuft der Datenweg, der verwendet wird, um die neuen Texturdaten in den Lokalspeicher in der Texturabbildungshardware zu laden, durch die Grundelementaufbereitungs-Pipeline des Systems. Wenn eine neue Textur abgebildet werden soll, muß folglich ermöglicht sein, daß die Grundelementaufbereitungs-Pipeline leer wird, bevor die neue Reihe von MIP-Tabellen heruntergeladen werden kann. Sobald die Reihe von MIP-Tabellen heruntergeladen ist, muß die Pipeline wiederum gefüllt werden. Die Notwendigkeit des Räumens der Grundelementaufbereitungs-Pipeline, jedesmal, wenn eine neue Textur erforderlich ist, reduziert die Bandbreite des Systems.

Stand der Technik

[0017] Die WO 97/24682 A betrifft ein System, welches einen Hauptspeicher und einen Cache-Speicher aufweist, von dem Texel-Daten in einen Interpolator eingegeben werden. Die beschriebenen Speicherelemente gehören entweder zu dem Hauptsystem oder zu dem Grafik-Teilsystem, wobei ein Haupt-Speicher des Grafik-Teilsystems einen Textur-Speicher (DRAM) und einen Cache-Speicher umfaßt, um Textur-Daten zwischenzuspeichern, um das Lesen und Schreiben der Texel zu beschleunigen. Zwischen dem Haupt-Speicher und dem Textur-Speicher ist kein weiterer Puffer vorgesehen.

[0018] Das "PC-Hardwarebuch" von Messmer, H.-P., Addison-Wesley, 1993, Seiten 233-240 beschäftigt sich mit der Funktionalität von PC-Hardware, und insbesondere mit den Betriebsprinzipien, die der Technologie von Cache-Speichern zugrundeliegen. Es wird ein allgemeiner Überblick über den Betrieb von Cache-Speichern in einem 8386-Prozessor gegeben.

Aufgabenstellung

[0019] Es ist die Aufgabe der vorliegenden Erfindung, ein Verfahren und eine Vorrichtung zu schaffen, die einen Betriebsmittelsparenden Zugriff auf Texturdaten mit einer hohen Systembandbreite in einem ermöglichen.

[0020] Diese Aufgabe wird durch ein Verfahren gemäß Anspruch 1 und durch ein System gemäß Anspruch 7 gelöst.

[0021] Bei einem Ausführungsbeispiel der Erfindung wird ein Verfahren zum Zugreifen auf Texturdaten in einem Texturabbildungs-Computergrafiksystem geschaffen. Das System weist einen Hostcomputer mit einem Hauptspeicher auf, der Texturdaten, die eine Mehrzahl von Texeln aufweisen, speichert. Das Verfahren weist folgende Schritte auf: Speichern zumindest eines Abschnitts der Texturdaten in einem Lokalspeicher des Systems; temporäres Speichern einer begrenzten Anzahl von Texeln, auf die in jüngster Zeit aus dem Lokalspeicher zugegriffen wurde, in einem Texeldatenpuffer; Lesen von Texeln von vordefinierten Orten des Texeldatenpuffers; und Zugreifen auf Texel von dem Lokalspeicher nur, wenn derartige Texel nicht verfügbar sind, um aus dem Texeldatenpuffer wiedergelesen zu werden.

[0022] Bei einem Ausführungsbeispiel weist das Verfahren ferner den Schritt des Bestimmens, für ein momentanes Texel, auf, ob aus dem Lokalspeicher auf das Texel zugegriffen wird, oder ob das Texel aus dem Texeldatenpuffer wiedergelesen wird. Der Schritt des Bestimmens weist den Schritt des Vergleichens einer Adresse des momentanen Texels mit einer Adresse eines Texels, auf das vorher zugegriffen wurde, auf.

[0023] Bei einem weiteren Ausführungsbeispiel der Erfindung wird ein Texturabbildungs-Computergrafiksystem geschaffen. Das System weist einen Hostcomputer mit einem Hauptspeicher auf, der Texturdaten, die eine Mehrzahl von Texeln einschließen, speichert. Ein Lokalspeicher speichert zumindest einen Teil der Texturdaten. Eine Lokalspeicher-Zugriffseinheit, die mit dem Lokalspeicher gekoppelt ist, greift auf Texel aus dem Lokalspeicher zu. Ein Texeldatenpuffer, der mit der Lokalspeicher-Zugriffseinheit gekoppelt ist, speichert eine begrenzte Anzahl von Texeln, auf die durch die Zugriffseinheit aus dem Lokalspeicher zuletzt zugegriffen wurde. Ein Texelinterpolator, der mit dem Texeldatenpuffer gekoppelt ist, liest Texel aus vorbestimmten Orten des Texeldatenpuffers. Die Zugriffseinheit greift auf Texel aus dem Lokalspeicher nur zu, wenn solche Texel nicht verfügbar sind, um aus dem Texeldatenpuffer wiedergelesen zu werden.

[0024] Bei einem Ausführungsbeispiel der Erfindung weist das System ferner eine Schaltung auf, und die mit dem Interpolator und der Lokalspeicher-Zugriffseinheit gekoppelt ist, die bestimmt, ob ein momentanes Texel verfügbar ist, um aus dem Texeldatenpuffer wiedergelesen zu werden. Bei einem

Ausführungsbeispiel weist das System ferner einen Texelinterpolator-Befehlspuffer auf, der mit dem Interpolator gekoppelt ist, welcher Befehle speichert, die anzeigen, aus welchen Orten in dem Texeldatenpuffer Texel gelesen werden sollen. Bei einem Ausführungsbeispiel weist das System ferner einen Texelzugriffs-Befehlspuffer auf, der mit der Lokalspeicher-Zugriffseinheit gekoppelt ist, welcher Befehle speichert, wobei jeder Befehl die Zugriffseinheit auffordert, auf ein Texel aus dem Lokalspeicher zuzugreifen.

[0025] Bei einem Ausführungsbeispiel weist der Lokalspeicher eine Mehrzahl von Verschachtelungen auf, und die Lokalspeicher-Zugriffseinheit weist eine Mehrzahl von Steuerungen auf, wobei eine Steuerung jeder Verschachtelung zugeordnet ist, wobei die Steuerungen parallel arbeiten, um getrennt auf Daten aus jeder Verschachtelung zuzugreifen. Bei diesem Ausführungsbeispiel weist das System ferner eine Mehrzahl von Texeldatenpuffern auf, wobei zumindest ein Texeldatenpuffer jeder Verschachtelung zugeordnet ist, wobei jede Steuerung getrennt auf ein Texel aus dem Lokalspeicher zugreift, nur wenn das Texel nicht verfügbar ist, um aus einem entsprechenden Texeldatenpuffer wiedergelesen zu werden.

Ausführungsbeispiel

[0026] Bevorzugte Ausführungsbeispiele der vorliegenden Erfindung werden nachfolgend bezugnehmend auf die beiliegenden Zeichnungen näher erläutert. Es zeigen:

[0027] **Fig.** 1 eine grafische Darstellung eines Satzes von Textur-MIP-Tabellen;

[0028] **Fig.** 2 ein Blockdiagramm eines Ausführungsbeispiels des gesamten Computergrafiksystems der vorliegenden Erfindung;

[0029] **Fig.** 2A ein Blockdiagramm eines weiteren Ausführungsbeispiels des gesamten Computergrafiksystems der vorliegenden Erfindung;

[0030] **Fig.** 3 ein Blockdiagramm der Texturabbildungshardware der vorliegenden Erfindung;

[0031] **Fig.** 4 ein detaillierteres Blockdiagramm des Parameterinterpolatorelements der Texturabbildungshardware der vorliegenden Erfindung;

[0032] **Fig.** 5 ein Blockdiagramm des Cache-Speichers und eines Teils der Texturabbildungshardware der vorliegenden Erfindung;

[0033] **Fig.** 6 ein Beispiel der Art und Weise, auf die Texturdatenblöcke organisiert sind, um einen Vorteil aus einer Vier-Verschachtelungs-Implementierung des Cache-Speichers der vorliegenden Erfindung zu ziehen;

[0034] **Fig.** 7 ein detailliertes Blockdiagramm der Organisation der Speicherchips, die den Cache-Speicher der vorliegenden Erfindung bilden;

[0035] **Fig.** 8 ein detailliertes Blockdiagramm eines Teils der Texturabbildungshardware der vorliegenden Erfindung;

[0036] **Fig.** 9 ein Diagramm und eine Grafik, die ein Beispiel von Texeln zeigen, auf die aus benachbarten MIP-Tabellen für jedes eines Stroms von Pixeln zugegriffen wird, gemäß einem Texturabbildungsschema der vorliegenden Erfindung;

[0037] **Fig.** 10 ein Diagramm von Texturabbildungs-Hardwarepuffern und zugeordneten Dateneinträgen gemäß dem Beispiel von **Fig.** 9;

[0038] **Fig.** 11 ein Blockdiagramm einer Schaltung, die durch die Texturabbildungshardware der vorliegenden Erfindung verwendet wird;

[0039] **Fig.** 12 ein Diagramm eines Beispiels eines Satzes von Textur-MIP-Tabellen;

[0040] **Fig.** 13 ein Diagramm, das zeigt, wie die MIP-Tabellen des Beispiels von **Fig.** 12 gemäß einem Speicher-Speicherungsschema der vorliegenden Erfindung in einem Speicher gespeichert sind;

[0041] **Fig.** 14 ein Diagramm einer MIP-Tabelle, das zeigt, wie die MIP-Tabelle gemäß einem Speicher-Speicherungsschema der vorliegenden Erfindung partitioniert ist;

[0042] **Fig.** 15 ein detaillierteres Diagramm von Teilen der Tabelle, die in **Fig.** 14 gezeigt ist, das zeigt, wie die Tabelle gemäß einem Speicher-Speicherungsschema der vorliegenden Erfindung weiter partitioniert ist;

[0043] **Fig.** 16 ein Diagramm, das die Art und Weise zeigt, auf die die Cache-Block-Kennung (cache block tag) erzeugt wird;

[0044] **Fig.** 17 ein Flußdiagramm, das ein Verfahren zum Bestimmen der Texeladresse mit einem entsprechenden Texturdatenblock aus mit Texeln versehenen, interpolierten Daten darstellt;

[0045] **Fig.** 18 ein Flußdiagramm, das ein Verfahren zum Bestimmen dessen darstellt, welcher Cache-Block ersetzt werden sollte, wenn ein Cache-Fehlzugriff (cache miss) auftritt;

[0046] **Fig.** 19 ein Diagramm, das die Texeltorregister zeigt, die in dem Texturabbildungschip vorgesehen sind;

[0047] **Fig.** 20 ein Flußdiagramm, das ein Verfahren zum Abwickeln von Cache-Fehlzugriffs-Unterbrechungen in dem Hostcomputer darstellt;

[0048] **Fig.** 21 ein Blockdiagramm des Cache-Mini-Verzeichnisses;

[0049] **Fig.** 22 ein Blockdiagramm des Cache-Hauptverzeichnisses;

[0050] **Fig.** 23 ein Blockdiagramm einer Reihe von Komparatoren, die vorgesehen sind, um Verhaltensnachteile zu reduzieren, wenn eine Cache-Lese-Kennung das Mini-Verzeichnis verfehlt; und

[0051] **Fig.** 24 ein Blockdiagramm einer beispielhaften Implementierung des Cache-Verzeichnisses der vorliegenden Erfindung.

I. Systemüberblick

[0052] **Fig.** 2 ist ein Blockdiagramm eines Ausführungsbeispiels eines Grafiksystems der vorliegenden Erfindung, das eine Texturabbildungshardware mit einem Cache-Speicher zum lokalen Speichern von Texturdaten aufweist. Es sollte offensichtlich sein, daß die veranschaulichende Implementierung nur exemplarisch bezüglich der Anzahl von Platinen und Chips, der Art und Weise, auf die dieselben partitioniert sind, der Busbreiten und der Datenübertragungsraten gezeigt ist. Zahlreiche andere Implementierungen können verwendet werden. Wie gezeigt ist, weist das System eine Eingangsplatine **10**, eine Texturabbildungsplatine **12** und eine Rahmenpufferplatine **14** auf. Die Eingangsplatine steht über einen 52-Bit-Bus **16** mit einem Hostcomputer **15** in Verbindung. Die Eingangsplatine empfängt Grundelemente, die aufbereitet werden sollen, über den Bus **16** von dem Hostcomputer. Die Grundelemente sind durch x-, y-, z-Vektorkoordinatendaten, R-, G-, B-Farbdaten und Texturkoordinaten S, T, alle für Abschnitte der Grundelemente, beispielsweise für die Scheitelpunkte, wenn das Grundelement ein Dreieck ist, spezifiziert. Daten, die die Grundelemente in drei Dimensionen darstellen, werden dann durch die Eingangsplatine **10** über einen 85-Bit-Bus **18** zu der Texturabbildungsplatine **12** und der Rahmenpufferplatine **14** geliefert. Die Texturabbildungsplatine interpoliert die empfangenen Grundelementdaten, um die Bildschirmanzeigepixel zu berechnen, die das Grundelement darstellen werden, und bestimmt die entsprechenden resultierenden Texturdaten für jedes Grundelementpixel. Die resultierenden Texturdaten werden über fünf 55-Bit-Busse **28**, die in **Fig.** 2 als ein einzelner Bus gezeigt sind, um die Figur deutlicher zu machen, zu der Rahmenpufferplatine geliefert.

[0053] Die Rahmenpufferplatine **14** interpoliert die Grundelementdaten, die von der Eingangsplatine **10** empfangen werden, ebenfalls, um die Pixel auf dem Anzeigebildschirm zu berechnen, die jedes Grundelement darstellen werden, und um die Objektfarbwerte für jedes Pixel zu bestimmen. Die Rahmenpufferplatine kombiniert dann auf einer Basis Pixel um Pixel die Objektfarbwerte mit den resultierenden Texturdaten, die von der Texturabbildungsplatine geliefert werden, um resultierende Werte R, G, B des Bilds für jedes Pixel zu erzeugen. R-, G-, B-Farb-Steuersignale für jedes Pixel werden jeweils über R-, G-, B-Leitungen **29** geliefert, um die Pixel des Anzeigebildschirms (nicht gezeigt) zu steuern, um auf dem Anzeigebildschirm ein resultierendes Bild anzuzeigen, das das mit einer Texturabbildung versehene Grundelement darstellt.

[0054] Die Eingangsplatine **10**, die Texturabbildungsplatine **12** und die Rahmenpufferplatine **14** sind jeweils pipelineartig aufgebaut und arbeiten auf mehreren Grundelementen gleichzeitig. Während die Texturabbildungs- und Rahmenpuffer-Platinen auf Grundelementen arbeiten, die vorher durch die Eingangsplatine geliefert werden, fährt die Eingangsplatine fort, auf neuen Grundelementen zu arbeiten und dieselben zu liefern, bis die Pipelines in den Platinen **12** und **14** voll werden.

[0055] Die Eingangsplatine **10** weist einen Verteilerchip **30**, drei Dreidimensionalgeometrie-Beschleunigerchips (3-D-Geometrie-Beschleunigerchips) **32A**, **32B** und **32C**, einen Zweidimensionalgeometrie-Beschleunigerchip (2-D-Geometrie-Beschleunigerchip) **34** und einen Konzentratorchip **36** auf. Der Verteilerchip **30** empfängt die x-, y-, z-Koordinaten- und Farb-Grundelementdaten über einen Bus **16** von dem Hostcomputer und verteilt 3-D-Grundelementdaten gleichmäßig unter den 3-D-Geometrie-Beschleunigerchips **32A**, **32B** und **32C**. Auf diese Art und Weise ist die Systembandbreite erhöht, da gleichzeitig drei Gruppen von Grundelementen bearbeitet werden. Daten werden über einen 40-Bit-Bus **38A** zu den 3-D-Geometrie-Beschleunigerchips **32A** und **32B** und über einen 40-Bit-Bus **38B** zu dem Chip **32C** geliefert. Beide Busse **38A** und **38B** übertragen Daten mit einer Rate von 60 MHz und liefern eine ausreichende Bandbreite, um zwei 3-D-Geometrie-Beschleunigerchips zu versorgen. 2-D-Grundelementdaten werden mit einer Rate von 40 MHz über einen 44-Bit-Bus **40** zu dem 2-D-Geometrie-Beschleunigerchip **34** geliefert.

[0056] Jeder 3-D-Geometrie-Beschleunigerchip transformiert die x-, y-, z-Koordinaten, die die empfangenen Grundelemente definieren, in entsprechende Bildschirmraumkoordinaten, bestimmt die R-, G-, B-Werte des Objekts und die S-, T-Werte der Textur für die Bildschirmraumkoordinaten, zerlegt Grundelementvierecke in Dreiecke und berechnet eine Dreieckflächengleichung, um jedes Dreieck zu definieren. Jeder 3-D-Geometrie-Beschleunigerchip führt

ferner Sichtausschnittoperationen durch, um eine genaue Bildschirmanzeige des resultierenden Bilds sicherzustellen, wenn mehrere Fenster angezeigt werden, oder wenn sich ein Abschnitt eines Grundelements über das Sichtvolumen, das auf dem Anzeigebildschirm dargestellt ist, hinaus erstreckt. Ausgangsdaten von den 3-D-Geometrie-Beschleunigerchips **32A**, **32B** bzw. **32C** werden mit einer Rate von 60 MHz jeweils über 44-Bit-Busse **42A**, **42B** und **42C** zu dem Konzentratorchip **36** geliefert. Der Zweidimensionalgeometrie-Beschleunigerchip **34** liefert ferner Ausgangsdaten über einen 46-Bit-Bus **44** mit einer Rate von 45 MHz zu dem Konzentratorchip **36**. Der Konzentratorchip **36** kombiniert die 3-D-Grundelementausgangsdaten, die von den 3-D-Beschleunigerchips **32A** bis **C** empfangen werden, sortiert die Grundelemente in die ursprüngliche Reihenfolge, die dieselben vor der Verteilung durch den Verteilerchip **30** hatten, um und liefert die kombinierten Grundelementausgangsdaten über den Bus **18** zu der Texturabbildungs- und der Rahmenpuffer-Platine.

[0057] Die Texturabbildungsplatine **12** weist einen Texturabbildungschip **46** und eine Lokalspeicher **48** auf, der vorzugsweise als ein Cache-Speicher ausgebildet ist. Bei einem bevorzugten Ausführungsbeispiel der Erfindung ist der Lokalspeicher aus Gründen, die nachfolgend erläutert werden, aus einer Mehrzahl von SDRAM-Chips (SDRAM = synchronous dynamic random access memory = synchroner dynamischer Direktzugriffsspeicher) gebildet. Wie nachfolgend detaillierter beschrieben wird, speichert der Cache-Speicher **48** Textur-MIP-Tabellendaten, die den Grundelementen zugeordnet sind, welche in der Rahmenpufferplatine aufbereitet werden. Die Textur-MIP-Tabellendaten werden über den Bus **40**, durch den 2-D-Geometrie-Beschleunigerchip **34** und über einen 24-Bit-Bus **24** aus einem Hauptspeicher **17** des Hostcomputers **15** heruntergeladen.

[0058] Der Texturabbildungschip **46** empfängt nacheinander Grundelementdaten über den Bus **18**, welche die Grundelemente darstellen, die auf dem Anzeigebildschirm aufbereitet werden sollen. Wie oben erläutert wurde, weisen die Grundelemente, die von den 3-D-Geometrie-Beschleunigerchips **32A** bis **C** geliefert werden, Punkte, Linien und Dreiecke auf. Die Texturabbildungsplatine führt keine Texturabbildung von Punkten oder Linien durch und bearbeitet nur Dreieck-Grundelemente. Die Daten, die die Dreieck-Grundelemente darstellen, enthalten die x-, y-, z-Pixelkoordinaten des Objekts für zumindest einen Scheitelpunkt, die R-, G-, B-Farbwerte des Objekts von zumindest einem Scheitelpunkt, die Koordinaten S, T der Abschnitte der Texturtabelle, die dem zumindest einem Scheitelpunkt entsprechen, und die Flächengleichung des Dreiecks. Der Texturabbildungschip **46** ignoriert die z-Koordinate des Objektpixels und die R-, G-, B-Farbwerte des Objekts. Der Chip **46** interpoliert die x-, y-Pixelkoordinaten und interpoliert die S- und T-Koordinaten, die jedem x-, y-Bildschirmanzeigepixel, das das Grundelement darstellt, entsprechen. Für jedes Pixel greift der Texturabbildungschip auf den Abschnitt der Textur-MIP-Tabelle, der demselben entspricht, aus dem Cache-Speicher zu und berechnet resultierende Texturdaten für das Pixel, welche einen gewichteten Durchschnitt mehrerer Texel einschließen können.

[0059] Bei einem exemplarischen Ausführungsbeispiel speichert der Cache vierundsechzig Blöcke von 256 × 256 Texeln. Anders als der Lokalspeicher, der in der Texturabbildungshardware bekannter Systeme verwendet ist, muß der Cache-Speicher der vorliegenden Erfindung nicht die gesamte Reihe von MIP-Tabellen der Textur speichern, die auf das Grundelement, das aufbereitet wird, abbildet, beispielsweise für große Texturen. Vielmehr speichert der Cache-Speicher zu jeder Zeit nur die speziellen Abschnitte der Reihe von MIP-Tabellen, die bei der gegenwärtigen Aufbereitung des Grundelements tatsächlich verwendet werden. Daher wird für die meisten Anwendungen nur ein Teil der gesamten Texturdaten für das Bild, das aufbereitet wird, zu jeder Zeit in dem Cache-Speicher gespeichert sein.

[0060] Die vollständige Reihe von MIP-Tabellen für jede Textur ist in dem Hauptspeicher **17** des Hostcomputers **15** angeordnet und gespeichert. Für jedes Pixel des Grundelements, das aufbereitet wird, greift der Texturabbildungschip **46** auf ein Verzeichnis des Cache-Speichers **48** zu, um zu bestimmen, ob das entsprechende Texel oder die Texel der Textur-MIP-Tabellen gegenwärtig in dem Cache vorliegen. Wenn die entsprechenden Texel zu der Zeit des Zugriffs in dem Cache-Speicher gespeichert sind, tritt ein Cache-Treffer auf, und die Texel werden aus dem Cache gelesen und durch den Texturabbildungschip **46** bearbeitet, um die resultierenden Texturdaten zu berechnen, die zu der Rahmenpufferplatine geleitet werden.

[0061] Wenn die entsprechenden Texel für das Grundelementpixel jedoch nicht in dem Cache-Speicher gespeichert sind, wenn ein Zugriff durch den Texturabbildungschip **46** stattfindet, tritt ein Cache-Fehlzugriff auf. Wenn ein Cache-Fehlzugriff auftritt, wird der Teil der Textur-MIP-Tabellendaten, der benötigt wird, um das Grundelement aufzubereiten, von dem Hauptspeicher **17** des Hostcomputers **15** in den Cache-Speicher **48** heruntergeladen, wobei möglicherweise einige Daten, die vorher in demselben gespeichert waren, ersetzt werden. Jedoch lädt anders als herkömmliche Texturabbildungssysteme, die die gesamte Reihe von MIP-Tabellen für jedes Grundelement, das aufbereitet wird, herunterladen, die vorliegende Erfindung nur den Teil der Reihe von MIP-Tabellen, der tatsächlich benötigt wird, um das Grundelement oder den momentan aufbereiteten Abschnitt desselben momentan aufzubereiten, herun-

ter. Wie detaillierter nachfolgend erläutert wird, wird, wenn ein Cache-Fehlzugriff stattfindet, ein Unterbrechungs-Steuersignal durch den Texturabbildungschip **46** erzeugt, um einen Texturunterbrechungsverwalter in dem Hostcomputer **15** zu initiieren. Das Unterbrechungs-Steuersignal wird über eine Leitung **94** zu dem Verteilerchip **30** geliefert, welcher wiederum ein Unterbrechungssignal über eine Leitung **95** zu dem Hostcomputer liefert.

[0062] Die angeforderten Texturdaten werden durch den Hostcomputer aus seinem Hauptspeicher wiedergewonnen und werden unter Umgehung der dreidimensionalen Grundelement-Aufbereitungspipeline durch die Eingangsplatine und den Texturabbildungschip über den Bus **24** zu der Texturabbildungsplatine **48** heruntergeladen. Wenn folglich eine Cache-Fehlzugriffs-Unterbrechung auftritt, kann die Eingangsplatine fortfahren, 3-D-Grundelemente zu bearbeiten und Ausgangsgrundelementdaten über den Bus **18** zu dem Texturabbildungschip und der Rahmenpufferplatine zu liefern, während die Texturdaten, die einem Grundelement zugeordnet sind, das den Cache-Fehlzugriff bewirkte, von dem Hauptspeicher **17** heruntergeladen werden. Im Gegensatz zu herkömmlichen Texturabbildungssystemen erfordert das Herunterladen von Texturdaten zu der Texturabbildungshardware kein Räumen der 3-D-Grundelementpipeline, wodurch die Bandbreite und das Verhalten des Systems verbessert werden. Die resultierenden Texturdaten für jedes Pixel werden über die fünf Busse **28** durch den Texturabbildungschip **46** zu der Rahmenpufferplatine geliefert. Die fünf Busse **28** sind jeweils mit fünf Rahmenpuffer-Steuerchips **50A**, **50B**, **50C**, **50D** und **50E**, die auf der Rahmenpufferplatine vorgesehen sind, gekoppelt und liefern parallel resultierende Texturdaten zu den Rahmenpuffer-Steuerchips. Die Rahmenpuffer-Steuerchips **50A** bis **E** sind jeweils mit Gruppen von zugeordneten VRAM-Chips **51A** bis **E** (VRAM = video random access memory = Video-Direktzugriffsspeicher) gekoppelt. Die Rahmenpufferplatine weist ferner vier Videoformatchips **52A**, **52B**, **52C** und **52D** und einen RAMDAW **54** auf (RAMDAW = Direktzugriffsspeicher-Digital/Analog-Wandler). Die Rahmenpuffer-Steuerchips steuern unterschiedliche, nicht überlappende Segmente des Anzeigebildschirms. Jeder Rahmenpuffer-Steuerchip empfängt Grundelementdaten über den Bus **18** von der Eingangsplatine und resultierende Texturabbildungsdaten über den Bus **28** von der Texturabbildungsplatine. Die Rahmenpuffer-Steuerchips interpolieren die Grundelementdaten, um die Bildschirmanzeige-Pixelkoordinaten in den jeweiligen Segmenten derselben, die das Grundelement darstellen, und die entsprechenden R-, G-, B-Farbwerte des Objekts für jede Pixelkoordinate zu berechnen. Für diejenigen Grundelemente (d.h. Dreiecke), für die resultierende Texturdaten von der Texturabbildungsplatine geliefert werden, kombinieren die Rahmenpuffer-Steuerchips auf einer Basis Pixel um Pixel die Objektfarbwerte und die resultierenden Texturdaten, um endgültige R-, G-, B-Werte für jedes Pixel, das auf dem Anzeigebildschirm angezeigt werden soll, zu erzeugen.

[0063] Die Art und Weise, auf die die Objekt- und Textur-Farbwerte kombiniert werden, kann auf eine Anzahl von unterschiedlichen Arten gesteuert werden. Beispielsweise können die Objektfarbwerte in einem Ersetzungsmodus einfach durch die Texturfarbwerte ersetzt werden, so daß nur die Texturfarbwerte beim Aufbereiten des Pixels verwendet werden. Alternativ können die Objekt- und Textur-Farbwerte in einem Modulationsmodus miteinander multipliziert werden, um die endgültigen R-, G-, B-Werte für das Pixel zu erzeugen. Außerdem kann ein Farbsteuerwort für jedes Texel gespeichert werden, das ein Verhältnis spezifiziert, das die Art und Weise definiert, auf die die entsprechenden Texturfarbwerte mit den Objektfarbwerten kombiniert werden sollen. Ein resultierendes Farbsteuerwort kann für die resultierenden Texeldaten, die jedem Pixel entsprechen, bestimmt werden und über den Bus **28** zu den Rahmenpuffer-Steuerchips geliefert werden, so daß die Steuerchips das Verhältnis verwenden können, das durch das entsprechende resultierende Steuerwort spezifiziert ist, um die endgültigen R-, G-, B-Werte für jedes Pixel zu bestimmen.

[0064] Die resultierenden Bildvideodaten, die durch die Rahmenpuffer-Steuerchips **50A** bis **E** erzeugt werden, einschließlich der R-, G-, B-Werte für jedes Pixel, werden in den entsprechenden VRAM-Chips **51A** bis **E** gespeichert. Jede Gruppe von VRAM-Chips **51A** bis E weist acht VRAM-Chips auf, derart, daß **40** VRAM-Chips auf der Rahmenpufferplatine angeordnet sind. Jeder der Videoformatchips **52A** bis **D** ist mit einem unterschiedlichen Satz von zehn VRAM-Chips verbunden und empfängt Daten von demselben. Die Videodaten werden seriell aus den VRAM-Chips geschoben und jeweils über 64-Bit-Busse **58A**, **58B**, **58C** und **58D** mit einer Rate von 33 Mhz zu den vier Videoformatchips **52A**, **52B**, **52C** und **52D** geliefert. Die Videoformatchips formatieren die Videodaten, so daß dieselben durch den RAMDAW gehandhabt werden können, und liefern die formatierten Daten über 32-Bit-Busse **60A**, **60B**, **60C** und **60D** mit einer Rate von 33 Mhz zu dem RAMDAW **54**. Der RAMDAW **54** wandelt die digitalen Farbdaten wiederum in analoge R-, G-, B-Farbsteuersignale um und liefert die R-, G-, B-Steuersignale für jedes Pixel entlang R-, G-, B-Steuerleitungen **29** zu einer Bildschirmanzeige (nicht gezeigt).

[0065] Bei einem Ausführungsbeispiel der Erfindung ist die Hardware auf der Texturabbildungsplatine **12** und der Rahmenpufferplatine **14** dupliziert, derart, daß bestimmte Grundelement-Aufbereitungsaufgaben auf mehreren Grundelementen parallel durchgeführt werden können, wodurch die Bandbreite des

Systems erhöht wird. Ein Beispiel eines solchen alternativen Ausführungsbeispiels der vorliegenden Erfindung ist in **Fig.** 2A gezeigt, welche ein Blockdiagramm eines Computergrafiksystems der vorliegenden Erfindung ist, bei dem ein bestimmter Teil der Hardware dupliziert ist. Das System von **Fig.** 2A weist vier 3-D-Geometrie-Beschleunigerchips **32A**, **32B**, **32C** und **32D**, zwei Texturabbildungschips **46A** und **46B**, denen jeweils Cache-Speicher **48A** und **48B** zugeordnet sind, und zehn Rahmenpufferchips **50A** bis **50J** auf, denen jeweils eine Gruppe von VRAM-Chips zugeordnet ist. Der Betrieb des Systems von

[0066] **Fig.** 2A ist gleichartig dem des Systems von **Fig.** 2, der oben beschrieben ist. Die Duplizierung der Hardware bei dem Ausführungsbeispiel von **Fig.** 2A ermöglicht eine erhöhte Systembandbreite, da bestimmte Grundelement-Aufbereitungsoperationen parallel auf mehreren Grundelementen durchgeführt werden können.

II. Übersicht: Texturabbildungschip

[0067] Ein Blockdiagramm des Texturabbildungschips **46** ist in **Fig.** 3 gezeigt. Der Chip **46** weist eine Eingangs-Pipelineschnittstelle **60** auf, die Objekt- und Textur-Grundelementdaten über den 64-Bit-Bus **18** von der Eingangsplatine empfängt. Die Dreieckgrundelemente, die durch den Texturabbildungschip bearbeitet werden, sind durch bis zu zweiundfünfzig digitale 32-Bit-Worte definiert, können jedoch durch Worte unterschiedlicher Längen definiert sein. Die Pipelineschnittstelle weist einen Satz von Master-Registern und einen Satz entsprechender Slave-Register auf. Während der Aufbereitung werden die Master-Register nacheinander mit den zweiundfünfzig digitalen Datenworten gefüllt, die das Grundelement definieren. Dann werden die Daten beim Empfang eines geeigneten Aufbereitungsbefehls in die Slave-Register in der Pipelineschnittstelle geschoben, was ermöglicht, daß die Master-Register in einer pipelineartigen Form mit Daten gefüllt werden, die ein weiteres Grundelement darstellen. Die Grundelementdaten, die über den Bus **18** geliefert werden, schließen die x-, y-, z-Vektorkoordinatendaten, die S-, T-Texturkoordinaten und die R-, G-, B-Objektfarbdaten für zumindest einen Dreiecksscheitelpunkt, ebenso wie Daten, die die Dreieckflächengleichung darstellen, ein. Wie oben erläutert wurde, ignoriert der Texturabbildungschip die z-Koordinate des Objektpixels und die R-, G-, B-Farbwerte des Objekts und speichert nur die anderen Daten in der Eingangs-Pipelineschnittstelle **60**.

[0068] Die Slave-Register der Pipelineschnittstelle **60** übertragen die Grundelementdaten über einen Bus **62** zu einer Parameter-Interpolatorschaltung **64**. Die Parameter-Interpolatorschaltung **64** interpoliert jedes Grundelementdreieck, um für jede Anzeigebildschirm-Pixelkoordinate, die das Dreieck darstellt, die S-, T-Texturtabellenkoordinaten für die Texturtabelle zu bestimmen, die auf das Pixel abbildet, sowie einen S- und T-Gradientenwert (Δ S, Δ T). Die S- und T-Gradienten sind jeweils gleich Änderungen der S- und T-Koordinaten zwischen benachbarten Pixeln und werden auf eine Art und Weise berechnet, die nachfolgend erläutert wird.

[0069] Die Parameter-Interpolatorschaltung **64**, die detaillierter in **Fig.** 4 gezeigt ist, weist ein Kantenschrittgeber **66**, einen FIFO-Puffer **68** (FIFO = "first-in, first-out" = zuerst hinein, zuerst heraus), einen Spannenschrittgeber **70** und eine Gradienten- und Perspektiven-Korrekturschaltung **72** auf, die alle seriell verbunden sind. Das Kantenschrittgeber startet bei der x-, y-Pixelkoordinate von einem der Dreieckscheitelpunkte und läuft unter Verwendung der Dreieckflächengleichung die Kanten des Dreiecks schrittweise ab, um die Pixelkoordinaten zu bestimmen, die die Dreieckkanten definieren. Für jede Pixelkoordinate werden Texturtabellenkoordinaten S und T basierend auf den S-, T-Werten der Dreieckscheitelpunkte bestimmt, um zu identifizieren, welche Texel in der Texturtabelle jeder Anzeigebildschirm-Pixelkoordinate entsprechen. Die Pixel- und Texelkoordinaten werden temporär in dem FIFO-Puffer gespeichert und dann zu dem Spannenschrittgeber geliefert. An jedem x-, y-Pixelort entlang einer Kante des Dreiecks läuft der Spannenschrittgeber die entsprechende Spanne des Dreiecks schrittweise ab, um die S-, T-Texelkoordinaten für jeden Pixelort entlang der Spanne zu bestimmen.

[0070] Jede S- und T-Koordinate für ein Anzeigebildschirmpixel kann einen ganzzahligen Teil und einen gebrochenen Teil aufweisen, wenn das Pixel nicht direkt (in einer Eins-Zu-Eins-Entsprechung) auf ein einzelnes Texel in einer der Reihe von MIP-Tabellen für die Textur abbildet. Wenn es auf die Texturtabelle abgebildet wird, kann, wie oben erklärt wurde, jedes Anzeigebildschirmpixel zwischen mehreren Texeln in einer der Reihe von MIP-Tabellen für die Textur liegen und kann ferner zwischen benachbarten (größenmäßig) MIP-Tabellen in der Reihe liegen.

[0071] Die Gradienten- und Perspektiven-Korrekturschaltung **72** bestimmt die Gradientenwerte von S und T (..S,..T) für jedes Anzeigebildschirmpixel. Bei einem Ausführungsbeispiel der Erfindung wird der Gradient Δ S ausgewählt, um größer zu sein als der Gradient Δ Sx und der Gradient Δ Sy, wobei der Gradient Δ Sx die Änderung der S-Koordinate in der Texturtabelle ist, wenn sich die Koordinate x zwischen benachbarten Pixeln auf dem Anzeigebildschirm ändert, und wobei der Gradient Δ Sy die Änderung der S-Koordinate ist, wenn sich die Koordinate y zwischen benachbarten Pixeln ändert. Der Gradient Δ T wird ähnlich berechnet. Die Gradienten Δ S, Δ T für ein Anzeigebildschirmpixel zeigen die Änderungsrate

der Koordinatenposition innerhalb der Texturtabelle für eine Änderung eines Pixels auf dem Anzeigebildschirm in der entsprechenden S-, T-Dimension an und werden verwendet, um zu bestimmen, auf welche MIP-Tabelle oder -Tabellen zugegriffen werden sollte, um die resultierenden Texturdaten für das Pixel zu liefern. Beispielsweise zeigt ein Gradient, der gleich zwei ist, für ein Anzeigebildschirmpixel an, daß das Pixel auf vier (d.h. **22**, wie nachfolgend erläutert wird) Texel abbildet, so daß auf die MIP-Tabelle, die bezüglich der Basistabelle größenmäßig um zwei reduziert ist (beispielsweise die Tabelle **102** in **Fig.** 1) zugegriffen werden sollte, um die resultierenden Texturdaten für das Pixel zu liefern. Folglich ist, wenn der Gradient zunimmt, die Größe der MIP-Tabelle, auf die zugegriffen wird, um die resultierenden Texturdaten für das Pixel zu liefern, reduziert.

[0072] Bei einem Ausführungsbeispiel der Erfindung wird ein einzelner Gradient, der gleich dem größeren von Δ S und Δ T ist, verwendet, um die geeignete MIP-Tabelle für jedes Pixel auszuwählen, derart, daß der Gradient gleich dem größten von Δ Sx, Δ Sy, Δ Tx und Δ Ty für das Pixel ist. Es sollte jedoch offensichtlich sein, daß der Gradient alternativ auf eine unterschiedliche Weise ausgewählt werden kann, beispielsweise durch das Auswählen des kleinsten dieser Werte, eines Durchschnitts dieser Werte oder einer bestimmten anderen Kombination. Da ein einzelner Gradient ausgewählt wird, der die Änderungsrate nur einer der S-, T-Koordinaten anzeigt, stellt das Quadrat des Gradienten die Anzahl von Texeln dar, die auf das entsprechende Pixel abbilden.

[0073] Der Parameterinterpolator bestimmt aus dem Gradienten die nächstliegende Tabelle, auf die das Pixel abbildet, und einen Wert, der anzeigt, wie stark das Pixel von dem direkten Abbilden auf diese Tabelle abweicht. Die nächstliegende Tabelle wird durch den ganzzahligen Teil einer Tabellennummer identifiziert, wobei der Wert, der anzeigt, um wieviel das Pixel von einer direkten Abbildung abweicht, durch eine gebrochene Komponente der Tabellennummer identifiziert ist.

[0074] Wie wiederum bezugnehmend auf das Blockdiagramm des Texturabbildungschips in **Fig.** 3 zu sehen ist, wird die Texeldatenausgabe von der Parameter-Interpolatorschaltung **64** über eine Leitung **70** zu einem Kachelbilder und Grenzprüfer **72** geliefert, der die Adresse der vier Texel bestimmt, die der Position in jeder der Texturtabellen, die durch die Texeldaten spezifiziert ist, am nächsten sind, und überprüft, um zu bestimmen, ob jede innerhalb der Grenze der Textur ist. Die Texeldaten weisen die interpolierten Koordinaten S, T (ganzzahlige und gebrochene Werte) ebenso wie die Tabellennummer und den Tabellenbruchteil auf. Der Kachelbilder verwendet den ganzzahligen Teil der Koordinaten S und T, die durch den Parameterinterpolator **64** berechnet sind, und addiert eins zu dem ganzzahligen Teil von jeder, um die Adressen der vier nächstliegeanden Texel zu erzeugen. Der Grenzprüfer bestimmt dann, ob die Koordinaten S, T für jedes dieser vier Texel aus der Grenze der Texturtabelle fallen. Wenn ein Anzeigebildschirmpixel auf eine S-, T-Koordinatenposition abbildet, die aus der Grenze der Texturtabelle fällt, ist eines von mehreren Texturabbildungsschematas implementiert, um zu bestimmen, ob irgendwelche resultierenden Texturdaten für dieses Pixel erzeugt werden müssen, und wie diese Daten erzeugt werden müssen. Beispiele derartiger Schemata schließen ein Umwickeln (wrapping) (eine Wiederholung der Textur), ein Spiegeln (ein Wiederholen des Spiegelbilds der Textur), ein Abschalten der Texturabbildung außerhalb der Grenze und ein Anzeigen einer soliden Farbe außerhalb der Grenze ein.

[0075] Die Fähigkeit des Zulassens, daß ein Pixel auf einen Ort in einer Texturtabelle abbildet, der jenseits der Grenze derselben liegt, liefert eine Flexibilität bezüglich der Art und Weise, auf die Texturen auf Objektgrundelemente abgebildet werden können. Es kann beispielsweise erwünscht sein, eine Textur in einer wiederholenden Form auf ein Objekt abzubilden, derart, daß die Textur auf mehrere Abschnitte des Objekts abgebildet wird. Wen eine Textur beispielsweise mit S-, T-Koordinaten im Bereich von [0, 0] einschließlich bis (10, 10) ausschließlich definiert ist, könnte ein Benutzer bestimmte Abschnitte des Objekts spezifizieren, um auf S-, T-Koordinaten [10, 10] einschließlich bis (20, 20) ausschließlich abzubilden. Die Schreibweise der in eckigen Klammern gesetzten Einschließlich-Koordinaten zeigt an, daß diese Koordinaten in dem Abschnitt der Textur, der auf das Objekt abgebildet wird, eingeschlossen sind, wohingegen das Objekt nur auf die S-, T-Koordinaten bis zu, jedoch ausschließlich der Ausschließlich-Koordinaten in runden Klammern abbildet. Wenn das Umwicklungsmerkmal für S-, T-Koordinaten ausgewählt ist, die aus der Grenze der Textur fallen, würden Pixel mit den S-, T-Koordinaten [10, 10] bis (20, 20) jeweils auf die Texel an den S-, T-Koordinaten [0, 0] bis (10, 10) abbilden.

[0076] Wie oben erläutert wurde können die resultierenden Texturdaten von einer zweidimensionalen Texturtabelle für ein einzelnes Pixel das Ergebnis einer Kombination von acht Texeln sein, d.h. den vier nächstliegenden Texeln in den zwei nächstliegenden MIP-Tabellen. Es gibt eine Anzahl von Arten, auf die die acht Texel kombiniert werden können, um die resultierenden Texeldaten zu erzeugen. Beispielsweise kann das einzige nächstliegende Texel in der nächstliegenden Tabelle ausgewählt werden, so daß keine Mittelung erforderlich ist. Alternativ können die einzigen nächstliegenden Texel in jeder der zwei nächstliegenden Tabellen basierend auf dem Wert des Gradienten miteinander gemittelt werden. Derartige Schemata bilden die Textur nicht so genau ab, wie

wenn die acht nächstliegenden Texel gemittelt werden.

[0077] Bei einem Ausführungsbeispiel der Erfindung wird eine trilineare Interpolation unterstützt, bei der die resultierenden Texturdaten für ein einzelnes Pixel als ein gewichteter Durchschnitt von acht Texeln berechnet werden können. Die den Gradienten darstellenden Änderungsraten von S, T werden verwendet, um die zwei nächstliegenden MIP-Tabellen zu identifizieren, aus denen auf Texturdaten zugegriffen werden soll, wobei auf die vier nächstliegenden Texel zugegriffen wird. Der Durchschnitt der vier Texel in jeder Tabelle wird basierend darauf, welche Texel am nächsten an den Koordinaten S, T der Position in der MIP-Tabelle, auf die das Anzeigebildschirmpixel abbildet, sind, gewichtet. Der gebrochene Teil der Koordinaten S und T für das Pixel wird verwendet, um diese Gewichtung durchzuführen. Der Durchschnittswert von jeder der zwei nächstliegenden MIP-Tabellen wird dann basierend auf dem Wert des Gradienten gewichtet. Ein Bruchwert wird zur Verwendung bei diesem Gewichtungsprozeß aus dem Gradienten berechnet. Beispielsweise liegt ein Gradient von drei mitten zwischen den MIP-Tabellen, die Gradienten von zwei bzw. vier entsprechen.

[0078] Das Texelinterpolationsverfahren wird durch die Texelinterpolatoren **76** durchgeführt. Die Bruchteile der Koordinaten S und T für jedes Anzeigebildschirmpixel werden von den Parameterinterpolatoren durch den Kachelbilder/Grenzprüfer über Leitungen **74** zu dem Texelinterpolator **76** geliefert. Die Bruchteile werden durch den Texelinterpolator verwendet, um die Gewichtung zu bestimmen, die jedem Texel während der Interpolation der mehreren Texel geboten wird, wenn resultierende Texeldaten berechnet werden.

[0079] Wie oben erläutert wurde, werden Textur-MIP-Tabellen, die einem Grundelement, das aufbereitet wird, zugeordnet sind, lokal in dem Cache-Speicher **48** (**Fig.** 2) gespeichert. Bei einem Ausführungsbeispiel der Erfindung ist der Cache voll assoziativ. Der Cache weist acht SDRAM-Chips auf, die in vier Verschachtelungen geteilt sind, wobei sich zwei SDRAM-Chips in jeder Verschachtelung befinden. Vier getrennte Steuerungen sind vorgesehen, wobei eine jeder Verschachtelung zugeordnet ist, so daß auf die SDRAM-Chips in jeder Verschachtelung gleichzeitig zugegriffen werden kann. Jeder SDRAM-Chip weist zwei verschiedene Speicherbänke auf, in denen bei aufeinanderfolgenden Lesezyklen auf verschiedene Speicherseiten zugegriffen werden kann, ohne unter Seitenumladungs-Nachteilen zu leiden, die üblicherweise dem Zugreifen auf Daten aus zwei unterschiedlichen Seiten (d.h. aus zwei unterschiedlichen Reihenadressen) in einem herkömmlichen DRAM zugeordnet sind.

[0080] Die Texturdaten (d.h. die MIP-Tabellen) sind in Texeldatenblöcke geteilt, die jeweils 256 × 256 Texel aufweisen. Der Cache-Speicher kann vierundsechzig Datenblöcke gleichzeitig speichern. Jeder Block weist eine zugeordnete Blockkennung (block tag) auf, die den Block eindeutig identifiziert. Der Cache weist ein Cache-Verzeichnis **78** auf, das die Blockkennungen speichert, die den Datenblöcken entsprechen, die gegenwärtig in dem Cache gespeichert sind. Wie nachfolgend detaillierter beschrieben wird, weist jede Blockkennung einen Texturidentifizierer (Textur-ID) auf, der die spezielle Textur identifiziert, die der Datenblock darstellt, eine Tabellennummer, die die spezielle MIP-Tabelle in der Reihe von Tabellen der Textur, die der Datenblock darstellt, identifiziert, und Koordinaten S und T hoher Ordnung, die den Ort des Datenblocks in der speziellen Tabelle identifizieren. Der physikalische Ort der Blockkennung in dem Cache-Verzeichnis stellt den Ort des entsprechenden Datenblocks in dem Cache-Speicher dar.

[0081] MIP-Tabellen von mehr als einer Textur können gleichzeitig in dem Cache-Speicher gespeichert sein, wobei der Texturidentifizierer zwischen den unterschiedlichen Texturen unterscheidet. Einige MIP-Tabellen enthalten weniger als 256 × 256 Texel und verbrauchen daher keinen ganzen Datenblock. Beispielsweise müssen die kleineren Tabellen in einer Reihe von MIP-Tabellen oder sogar die größeren Tabellen für kleine Texturen nicht 256 × 256 Texel übersteigen. Um den Speicherraum effizient zu nutzen können Abschnitte mehrerer Tabellen in einem einzelnen Texturdatenblock gespeichert werden, wobei jeder Tabellenabschnitt einem Unterblock in dem Block zugewiesen ist. Jede der mehreren Tabellen, die in einem einzelnen Block gespeichert sind, weist einen zugeordneten Untertexturidentifizierer (ID) auf, der den Ort der Tabelle in dem Block identifiziert.

[0082] Während der Aufbereitung erzeugt der Kachelbilder/Grenzprüfer 72 eine Lese-Cache-Kennung für den Texturdatenblock, der auf das Pixel, das aufbereitet werden soll, abbildet. Die Art und Weise, auf die die Kennungen erzeugt werden, ist nachfolgend detaillierter erläutert. Die Kennungen sind 23-Bit-Felder, die acht Bits, die die Textur-ID der Texturdaten darstellen, ein Bit, das beim Bestimmen der Tabellennummer der Texturdaten verwendet wird, und die sieben Koordinaten S und T hoher Ordnung der Texturdaten aufweisen. Der Cache-Speicher **78** vergleicht die Lese-Cache-Kennung, die von dem Kachelbilder/Grenzprüfer geliefert wird, mit den Blockkennungen, die in dem Verzeichnis gespeichert sind, um zu bestimmen, ob der Texturdatenblock, der beim Aufbereiten verwendet werden soll, in dem Cache-Speicher ist. Wenn die Blockkennung der Texturdaten, die auf das Grundelement abbilden, das aufbereitet werden soll, in dem Cache-Verzeichnis gespeichert ist (d.h. trifft), erzeugt das Cache-Verzeichnis einen Blo-

ckindex, der den physikalischen Ort des Texturdatenblocks in dem Cache anzeigt, der der Trefferkennung entspricht. Die Berechnung des Blockindex wird nachfolgend detaillierter erläutert. Ferner wird eine Texeladresse durch den Kachelbilder/Grenzprüfer **72** für jedes Texel, das aus dem Cache gelesen werden soll, erzeugt, die den Ort des Texels in dem Block anzeigt. Die Texeladresse weist Adressbits niedriger Ordnung der interpolierten Koordinaten S, T für größere Tabellen auf und wird basierend auf einem Algorithmus berechnet, der nachfolgend für kleinere Tabellen beschrieben wird. Der Blockindex und die Texeladresse weisen zusammen die Cache-Adresse auf, die den Ort des Texels in dem Cache anzeigt. Wie nachfolgend detaillierter erläutert wird, werden die LSBs (LSB = least significant bit = niederstwertiges Bit) der Koordinaten S und T für jedes Texel decodiert, um zu bestimmen, in welcher der vier Cache-Verschachtelungen das Texel gespeichert ist, wobei die übrigen Bits der Cache-Adresse zu der Texel-Cache-Zugriffsschaltung **82** zusammen mit einem Befehl über eine Leitung **84** geliefert werden, um die Texeldaten, die an dem adressierten Ort in dem Cache gespeichert sind, zu lesen.

[0083] Wenn die Lese-Cache-Kennung nicht mit irgendeiner der Blockkennungen, die in dem Cache-Verzeichnis **78** gespeichert sind, übereinstimmt, tritt ein Fehlzugriff auf und das Cache-Verzeichnis **78** erzeugt ein Unterbrechungs-Steuersignal über eine Leitung **94** (**Fig.** 2) zu dem Verteilerchip **30** auf der Eingangsplatine, welcher über eine Leitung **95** eine Unterbrechung zu dem Hostcomputer **15** erzeugt. Als Reaktion auf die Unterbrechung führt der Prozessor **19** des Hostcomputers eine Dienstroutine durch, die nachfolgend detaillierter erläutert wird, die die Fehlzugriff-Blockkennung von dem Cache-Verzeichnis liest und den entsprechenden Texturdatenblock auf eine Art und Weise, die die 3-D-Grundelementpipeline in der Eingangsplatine **10** und dem Texturabbildungschip **46** umgeht, in den Cache-Speicher herunterlädt. Die Texturdaten, die aus dem Hauptspeicher heruntergeladen werden, werden über den Bus **24**, durch den Texeltor **92** (**Fig.** 3) zu der Texel-Cache-Zugriffsschaltung **82** geliefert, die die Daten in die SDRAMs schreibt, die den Cache-Speicher bilden.

[0084] Wenn ein Cache-Fehlzugriff auftritt, wartet der Texturabbildungschip darauf, daß die nächsten Texturdaten heruntergeladen werden, bevor derselbe mit der Verarbeitung des Grundelements fortfährt, bei dem der Fehlzugriff auftrat. Jedoch fahren die Stufen der Pipeline, die dem Cache-Lesen folgen, fort, diejenigen Grundelemente zu verarbeiten, die vor dem Fehlzugriff-Grundelement empfangen wurden. In gleicher Weise fahren auch die Stufen der Pipeline, die dem Cache-Lesen vorgeschaltet sind, fort, Grundelemente zu verarbeiten, es sei denn und bis die Pipeline nach der Cache-Leseoperation voll wird, während sie auf das Herunterladen der neuen Texturdaten wartet.

[0085] Während der Aufbereitung fahren die hinteren Stufen der Pipeline in der Rahmenpufferplatine **14** nicht mit der Verarbeitung eines Grundelements fort, bis die Texturdaten, die dem Grundelement entsprechen, von der Texturabbildungsplatine empfangen werden. Daher wartet, wenn ein Cache-Fehlzugriff auftritt und der Texturabbildungschip darauf wartet, daß neue Texturdaten heruntergeladen werden, die Rahmenpufferplatine **14** in gleicher Weise darauf, daß die resultierenden Texturdaten von dem Texturabbildungschip geliefert werden. Wie bei dem Texturabbildungschip fahren auch die Stufen der Pipeline, die der Stufe folgen, die die Texturabbildungsdaten empfangen, fort, diejenigen Grundelemente zu verarbeiten, die vor dem Fehlzugriff-Grundelement empfangen wurden, und die Stufen der Pipeline, die der Stufe, die Texturabbildungsdaten empfängt, vorgeschaltet sind, fahren ebenfalls fort, Grundelemente zu verarbeiten, es sei denn und bis die Pipeline voll wird.

[0086] Es sollte offensichtlich sein, daß, wenn die Pipeline entweder der Texturabbildungsplatine oder der Rahmenpufferplatine einen Wartezustand annimmt, wenn sie als Reaktion auf einen Cache-Fehlzugriff auf neue Texturdaten wartet, die Pipeline in der Eingangsplatine **10** in gleicher Weise einen Wartezustand annimmt. Da Cache-Fehlzugriffe auftreten werden und einen Zugriff auf den Hauptspeicher des Hostcomputers und ein Herunterladen von Texturdaten, dessen Vollendung mehrere Zyklen benötigen wird, zur Folge haben werden, ist es erwünscht, sicherzustellen, daß die Pipeline in dem Texturabbildungschip niemals infolgedessen warten muß, daß die Pipeline in der Rahmenpufferplatine einen Wartezustand angenommen hat. Daher ist die Rahmenpufferplatine bei einem Ausführungsbeispiel der Erfindung mit einer tieferen Grundelementpipeline versehen als die Texturabbildungsplatine, so daß die Texturabbildungsplatine nicht durch das Warten darauf, daß die Rahmenpufferplatine verfügbar wird, verzögert werden sollte.

[0087] Bei einem Ausführungsbeispiel der Erfindung ist die Fähigkeit geliefert, die Texturabbildung auszuschalten. Dies wird durch einen Softwarebetrieb auf dem Prozessor **19** des Hostcomputers erreicht, um ein Register sowohl in der Texturabbildungsplatine **12** als auch in der Rahmenpufferplatine **14** einzustellen. Wenn sie eingestellt sind, um die Texturabbildung abzuschalten, hindern diese Register jeweils den Texturabbildungschip **46** daran, Texturdaten zu der Rahmenpufferplatine **14** zu liefern, und weisen die Rahmenpufferplatine an, mit dem Aufbereiten von Grundelementen fortzufahren, ohne auf Texturdaten von der Texturabbildungsplatine zu warten.

[0088] Wie oben beschrieben wurde kann für jedes Anzeigebildschirmpixel, das mit Texturdaten von einer zweidimensionalen Texturtabelle aufbereitet wird, auf vier Texel aus einer MIP-Tabelle (bilineare Interpolation) oder acht Texel aus zwei benachbarten MIP-Tabellen (trilineare Interpolation) aus dem Cache-Speicher zugegriffen werden, um die resultierenden Texturdaten für das Pixel zu bestimmen. Die Texel, die aus dem Cache gelesen werden, werden über einen Bus **86** (**Fig.** 3) zu dem Texelinterpolator **76** geliefert, der die mehreren Texel interpoliert, um resultierende Texeldaten für jedes Pixel zu berechnen. Die Interpolation kann abhängig von einem Modus, der für das System eingerichtet ist, variieren. Wenn ein Punktabtast-Interpolationsmodus eingerichtet ist, sind die resultierenden Texeldaten gleich dem einzelnen Texel, das am nächsten an dem Ort ist, der durch die Koordinaten S, T des Pixels in der Texturtabelle definiert ist. Wenn alternativ eine bilineare oder eine trilineare Interpolation verwendet ist, sind die resultierenden Texeldaten jeweils ein gewichteter Durchschnitt der vier oder acht nächstliegenden Texel in der einen oder den zwei nächstliegenden Tabellen. Die Gewichtung, die jedem der mehreren Texel gegeben wird, wird basierend auf dem Wert des Gradienten und den Bruchkomponenten der Koordinaten S und T, die dem Texelinterpolator **76** von dem Kachelbilder/Grenzprüfer geliefert werden, bestimmt.

[0089] Die resultierenden Texeldaten für die Anzeigebildschirmpixel werden sequentiell über einen Bus **88** zu einem Rahmenpufferschnittstellen-FIFO-Puffer **90** geliefert. Der Rahmenpufferschnittstellen-FIFO-Puffer **90** kann bis zu vierundsechzig resultierende Texel speichern.

[0090] Jedes resultierende Texel ist ein 32-Bit-Wort, das acht Bits aufweist, um sowohl R, G, B als auch α darzustellen. Das Byte α zeigt der Rahmenpufferplatine **14** (**Fig.** 2) die Art und Weise an, auf die die Werte R, G, B der resultierenden Texturdaten mit den Werten R, G, B der Objektdaten, die durch die Rahmenpufferplatine beim Berechnen der endgültigen Anzeigebildschirmwerte R, G, B für jedes Pixel, das auf das Texel abbildet, erzeugt werden, kombiniert werden sollen. Die Ausgaben T0 bis T4 des Rahmenpufferschnittstellen-FIFO--Puffers werden über den Bus **28** zu der Rahmenpufferplatine **14** (**Fig.** 2) geliefert. Die Rahmenpufferplatine kombiniert die Werte R, G, B der resultierenden Texeldaten mit den Objektwerten R, G, B auf die Art und Weise, die durch α spezifiziert ist, um endgültige Werte R, G, B für jedes Anzeigebildschirmpixel zu erzeugen.

III. Cache-Speicherorganisation

[0091] **Fig.** 5 ist ein Blockdiagramm einer Cache-Speicherimplementierung gemäß einem darstellenden Ausführungsbeispiel der vorliegenden Erfindung, die mit Abschnitten des Texturabbildungschips, die das Texeltor **92**, den Texturinterpolator **76**, das Cache-Verzeichnis **78** und die Texel-Cache-Zugriffsschaltung **82** einschließen, gekoppelt ist. Bei diesem darstellenden Ausführungsbeispiel weist der Cache-Speicher **48** vier Verschachtelungen **204A**, **204B**, **204C** und **204D** auf. Jede Verschachtelung weist zwei SDRAM-Chips (nicht gezeigt) auf, auf die gleichzeitig zugegriffen werden kann, von denen jeder während eines Lesezyklus acht Datenbits liefert. Daher liefert jede Verschachtelung sechzehn Texeldatenbits während eines einzelnen Lesezyklus. Jedes 32-Bit-Texeldatenwort wird in einer einzelnen Verschachtelung in dem Cache gespeichert, wobei acht Bit in jedem von zwei aufeinanderfolgenden Orten in jedem SDRAM in der Verschachtelung gespeichert werden. Um ein Texel aus dem Cache zu lesen, werden folglich zwei Lesezyklen an aufeinanderfolgenden Orten in der entsprechenden Verschachtelung durchgeführt, um die zweiunddreißig Texeldatenbit zu liefern. Wie nachfolgend erklärt wird muß nur ein Adreßwort (einschließlich der Zeilen- und Spalten-Daten) zu den SDRAMs in jeder Verschachtelung geliefert werden, um bei zwei aufeinanderfolgenden Zyklen ein Datenbündel zu ergeben. Das Bündel weist sechzehn Bit, die bei einem ersten Zyklus von der gegebenen Adresse geliefert werden, und sechzehn Bit, die bei einem zweiten Zyklus von einer Adresse mit der gleichen Zeile und einer Spalte, die um eins inkrementiert ist, geliefert werden, auf.

[0092] Die Texel-Cache-Zugriffsschaltung **82** weist ferner vier getrennte Steuerungen auf, die mit Steuerung A (**200A**), Steuerung B (**200B**), Steuerung C (**200C**) und Steuerung D (**200D**) bezeichnet sind. Die vier Steuerungen A, B, C und D können durch parallele Busse **202A**, **202B**, **202C** und **202D** gleichzeitig auf die vier Verschachtelungen **204A**, **204B**, **204C** und **204D** zugreifen. Die Steuerungen lesen als Reaktion auf Befehle und an Adressen, die jeweils über Busse **84A**, **84B**, **84C** und **84D** empfangen werden, Texeldaten aus dem Speicher **48**.

[0093] Wie oben beschrieben wurde, kann jedes Pixel potentiell auf vier Texel aus einer MIP-Tabelle abbilden, oder auf acht Texel aus mehreren MIP-Tabellen. Wie nachfolgend detaillierter erläutert wird, sind Texeldaten, die in den Cache heruntergeladen werden, in dem Hauptspeicher des Hostcomputers derart organisiert, daß beliebige vier benachbarte Texel in jeder MIP-Tabelle in getrennten Verschachtelungen angeordnet sind, so daß parallel auf dieselben zugegriffen werden kann. Folglich können alle vier benachbarten Texel in einer MIP-Tabelle, auf die möglicherweise zugegriffen werden muß, um durch eine bilineare Interpolation resultierende Texeldaten zu erzeugen, in einer einzelnen Leseoperation gelesen werden. Wenn eine trilineare Interpolation verwendet ist, können die zwei Sätze von vier Texeln aus benachbarten MIP-Tabellen in zwei Leseoperati-

onen gelesen werden.

[0094] **Fig.** 6 zeigt ein Beispiel der Art und Weise, auf die Texturdatenblöcke (wobei nur einige Texel gezeigt sind) organisiert sind, um aus der Implementierung mit vier Verschachtelungen des Cache-Speichers, die ermöglicht, daß alle vier benachbarten Texel in einer MIP-Tabelle gleichzeitig gelesen werden, einen Vorteil zu ziehen. Jedes Texel ist mit A, B, C oder D bezeichnet, um die Verschachtelung in dem Cache-Speicher zu identifizieren, in der das Texel gespeichert ist. Das Muster der Bezeichnungen A bis D wiederholt sich, so daß jeder Ort in der Tabelle zwischen vier Texel fällt, die mit A, B, C und D bezeichnet sind. Folglich sind für ein Pixel, das auf irgendeinen Ort in der Tabelle abbildet, die vier nächstliegenden Texel in getrennten Verschachtelungen A bis D, derart, daß durch die vier unabhängigen Steuerungen **200A** bis **D** gleichzeitig auf dieselben zugegriffen werden kann. Beispielsweise bildet das Pixel P0 auf einen Ort zwischen vier Texeln, die mit A, B, C und D bezeichnet sind, ab, während das Pixel P1 auf einen Ort zwischen vier Texeln, die mit B, A, D und C bezeichnet sind, abbildet.

[0095] Es sollte offensichtlich sein, daß die oben beschriebene Cache-Implementierung nur zu darstellenden Zwecken geboten wurde, und daß andere Implementierungen verwendet sein können. Beispielsweise kann der Cache mit acht getrennten Verschachtelungen mit acht getrennten Steuerungen implementiert sein, derart, daß, wenn eine trilineare Interpolation verwendet ist, in einer einzelnen Leseoperation gleichzeitig auf acht Texel aus dem Cache zugegriffen werden kann. Jeder SDRAM-Chip in dem Cache-Speicher ist intern in zwei gleich große Bänke geteilt, die gleichzeitig getrennte aktive Seiten halten können (d.h. Gruppen von Speicherorten mit einer gemeinsamen Zeilenadresse). Folglich kann bei aufeinanderfolgenden Lesezyklen auf Daten aus unterschiedlichen Seiten in den zwei Bänken eines SDRAM-Chips zugegriffen werden, ohne den Nachteil eines Seitenumladens zu erleiden, der gewöhnlich dem aufeinanderfolgenden Lesen von Daten aus unterschiedlichen Seiten in einem herkömmlichen DRAM zugeordnet ist.

[0096] Wie nachfolgend detaillierter erläutert wird, sind die Texturdaten in dem Cache-Speicher organisiert, um aus diesem Merkmal der SDRAMs einen Vorteil zu ziehen, um Seitenkreuzungsnachteile zu minimieren, wenn eine trilineare Interpolation durchgeführt wird. Die acht Texel, die für eine trilineare Interpolation erforderlich sind, weisen Sätze von vier Texeln aus zwei MIP-Tabellen auf. Jeder Satz von vier benachbarten Texeln in einer einzelnen Tabelle ist derart angeordnet, daß auf die Art und Weise, die oben beschrieben wurde, einer in jeder der Verschachtelungen A, B, C und D gespeichert ist, so daß gleichzeitig auf die vier Texel zugegriffen werden kann. Außerdem sind gemeinsame Daten von benachbarten MIP-Tabellen in der Reihe von Tabellen für jede Textur in unterschiedlichen SDRAM-Banken in dem Cache gespeichert. Wenn eine trilineare Interpolation durchgeführt wird, werden vier Texel aus einer MIP-Tabelle gleichzeitig aus einer der SDRAM-Bänke von Verschachtelungen A bis D während der zwei Lesezyklen eines ersten Bündels gelesen, während vier Texel aus einer benachbarten MIP-Tabelle aus der anderen SDRAM-Bank während der zwei Lesezyklen eines nachfolgenden Bündels gelesen werden. Da beide Bänke von SDRAMs gleichzeitig Zeilen-aktiv sein können, kann in Rücken-an-Rücken-Bündeln auf die zwei Sätze von vier Texeln zugegriffen werden, ohne einen Seitenumladungsnachteil zu erleiden. Es sollte offensichtlich sein, daß, wenn Pixel eines Objekts aufbereitet werden, benachbarte Pixel häufig auf die gleichen zwei MIP-Tabellen für die Textur abbilden, was erfordert, daß Cache-Lesevorgänge kontinuierlich zwischen den Cache-Blöcken, die die gemeinsamen Daten in den zwei Tabellen speichern, geschaltet werden. Die Cache-Organisation der vorliegenden Erfindung, die ermöglicht, daß zwei Seiten in jedem SDRAM aktiv bleiben, ist vorteilhaft, da sie ermöglicht, daß während der Aufbereitung von Anzeigebildschirmpixeln eine trilineare Interpolation durchgeführt wird, ohne bei jedem Zyklus, wenn zwischen zwei benachbarten MIP-Tabellen geschaltet wird, einen Seitenumladungsnachteil zu erleiden.

[0097] **Fig.** 7 ist ein detaillierteres Blockdiagramm der oben beschriebenen, darstellenden Implementierung des Cache-Speichers der vorliegenden Erfindung. Der Cache weist acht SDRAM-Chips, die mit SD1 bis SD8 bezeichnet sind, auf, die gleichmäßig unter den vier Verschachtelungen **204A** bis **204D** verteilt sind, wobei jede Verschachtelung zwei SDRAM-Chips aufweist. Die zwei SDRAMs in jeder Verschachtelung verwenden die folgenden Leitungen gemeinsam: elf Adreßleitungen (ADD), Zeilen- und Spalten-Adreß-Übernahmesignale (RAS und CAS), eine Schreibfreigabe (WE), eine Taktfreigabe (CKE) und eine Dateneingangs/Ausgangsmaske (DQM). Die SDRAMs in jeder Verschachtelung sind mit acht getrennten Datenleitungen gekoppelt, durch die acht Datenbits während jedes Lese- oder Schreib-Zyklusses gelesen bzw, geschrieben werden. Jeder SDRAM-Chip weist zwei Speicherbänke auf, wobei jede Bank 1.048.576 8-Bit-Texturdatenworte speichert.

[0098] Auf die zwei SDRAMs in jeder Verschachtelung kann gleichzeitig zugegriffen werden, wobei dieselben zusammen sechzehn Datenbits liefern, wobei der eine der SDRAMs Datenbits [15:08] liefert, während der andere Datenbits [07:00] liefert. Wie oben erläutert wurde, ergeben zwei aufeinanderfolgende Lesezyklen eines einzelnen Bündels ein vollständiges 32-Bit-Datentexel von jeder Verschachtelung mit

einem getrennten 8-Bit-Wort, das jeden der Werte R, G, B und α für das Texel darstellt.

[0099] Die SDRAM-Chips empfangen zwanzig Adressbits, die auf den elf Adreßleitungen ADD multiplext werden, um die 1.048.576 8-Bit-Worte in jeder Bank zu decodieren. Wie nachfolgend detailliert erklärt wird, werden ein 6-Bit-Blockindex und eine 16-Bit-Texeladresse für jedes Texel, auf das aus dem Cache zugegriffen werden soll, berechnet. Der Blockindex zeigt an, in welchem der vierundsechzig Datenblöcke sich das Texel befindet, während die Texeladresse die präzise S-, T-Koordinatenadresse des Texels in dem Block anzeigt. Die Texeladresse weist acht S-Bits und acht T-Bits auf, unter der Annahme eines quadratischen Datenblocks, der 256 × 256 Texel aufweist. Eine Cache-Adresse ist ein 22-Bit-Wort, das die Kombination des Blockindex (sechs MSBs) (MSB = most significant bit = höchstwertiges Bit) und der Texeladresse (sechzehn LSBs) aufweist. Die Cache-Adresse zeigt den exakten Ort des Texels in dem Cache an.

[0100] Während der Aufbereitung decodiert der Kachelbilder/Grenzprüfer das LSB-S-Bit und das LSB-T-Bit der Texeladresse (d.h. die LSB-S-Koordinate und die LSB-T-Koordinate), um zu bestimmen, in welchen der vier Verschachtelungen des Caches das Texel gespeichert ist. Die verbleibenden zwanzig größeren Adreßbits der Cacheadresse werden entlang der Adreßleitungen ADD zu den zwei SDRAM-Chips in der entsprechenden Verschachtelung geliefert. Von den zwanzig Adreßbits, die zu den zwei SDRAMs geliefert werden, werden neun Bit verwendet, um die Spalte auszuwählen, und elf Bit werden verwendet, um die Zeile in den SDRAMs auszuwählen, um auf die Texeldaten zuzugreifen. Für Fachleute sollte es offensichtlich sein, daß die Spalten- und Zeilen-Adreßbits separat bei unterschiedlichen Zyklen in die SDRAMs zwischengespeichert werden, wobei die RAS- und CAS-Übernahmesignale herkömmlich verwendet werden, um auf die Daten zuzugreifen.

[0101] Während eines Zweizyklus-Bündels werden sechzehn Bit von dem adressierten Ort der zwei SDRAMs in der gleichen Verschachtelung während des ersten Zyklusses geliefert, und dann werden, ohne das Liefern einer weiteren Adresse, sechzehn Bit von einem anderen Ort der zwei SDRAMs während des zweiten Zyklusses geliefert. Die Adresse bei dem zweiten Zyklus weist die gleiche Zeilenadresse auf und eine Spaltenadresse, die um eins inkrementiert ist. Es sollte ferner offensichtlich sein, daß, sobald eine Seite (spezielle Zeilenadresse) aktiviert ist, dieselbe aktiviert bleibt, bis eine andere Zeilenadresse geliefert wird. Wenn sich aufeinanderfolgende Texel, auf die aus der gleichen Verschachtelung zugegriffen werden soll, auf der gleichen Seite befinden (die gleiche Zeilenadresse aufweisen), muß daher die Zeilenadresse nur einmal während des ersten der aufeinanderfolgenden Bündel geliefert werden.

[0102] Ferner sind die Leitungen RAS, CAS und WE verwendet, um Daten auf eine herkömmliche Art und Weise zu dem SDRAM-Chip zu adressieren und in denselben zu schreiben. Wenn das Taktfreigabesignal CKE deaktiviert ist, ist der interne Takt angehalten. Die SDRAMs sprechen durch das Intakthalten der Daten, wobei beide Bänke in einen Wartezustand gesetzt werden, auf dieses Signal an. Das Dateneingabe/Ausgabe-Maskensignal DQM wirkt als eine Ausgabefreigabe während eines Lesezyklusses und als eine Eingabe-Datenmaske während eines Schreibzyklusses.

[0103] Die SDRAMs werden herkömmlich verwendet, indem bestimmt wird, von welcher zukünftigen Seite auf nachfolgende Daten zugegriffen werden wird, während von einer momentanen Seite auf gegenwärtige Daten zugegriffen wird, und indem die zukünftige Seite aktiviert wird, bevor der Lesezyklus der gegenwärtigen Daten abgeschlossen ist. Da SDRAMs ermöglichen, daß zwei unterschiedliche Seiten gleichzeitig aktiv sind, vermeidet die herkömmliche Verwendung der SDRAMs Seitenumladungsnachteile, die gewöhnlich dem Zugreifen auf Daten von unterschiedlichen Seiten in herkömmlichen DRAMs zugeordnet sind. Die Verwendung herkömmlicher SDRAMs liefert jedoch diesen Vorteil nicht, wenn sich Daten, die bei vielen aufeinanderfolgenden Lesezyklen gelesen werden sollen, auf unterschiedlichen Seiten befinden, da mehr als ein Zyklus erforderlich ist, um vorauszuschauen und eine zukünftige Seite zu aktivieren. Das Texturdaten-Speicherverfahren der vorliegenden Erfindung liefert einen Vorteil gegenüber der herkömmlichen Verwendung des SDRAM, indem ermöglicht wird, daß eine Vielzahl aufeinanderfolgender SDRAM-Lesezyklen von unterschiedlichen Seiten ohne das Erleiden eines Nachteils stattfinden. Speziell durch das Speichern gemeinsamer Daten von benachbarten MIP-Tabellen einer Textur (auf die während aufeinanderfolgender Lesezyklen zugegriffen werden muß, wenn eine trilineare Interpolation ausgeführt wird) in getrennten Bänken der SDRAMs kann ohne einen Nachteil bei aufeinanderfolgenden Lesezyklen auf die Daten aus den unterschiedlichen Bänken zugegriffen werden. Obwohl das Verfahren der vorliegenden Erfindung der Datenspeicherzuweisung zum Verbessern des SDRAM-Verhaltens bezüglich der Speicherung von Texturabbildungsdaten zeigt und beschrieben wurde, sollte es offensichtlich sein, daß das Verfahren der vorliegenden Erfindung nicht derart begrenzt ist. Speziell ist das Verfahren anwendbar, um alle Datentypen zuzuweisen, bei denen mehrere aufeinanderfolgende Lesezyklen auf Daten aus unterschiedlichen Speicherorten zugreifen.

IV. Cache-Steuer-FIFOs

[0104] **Fig.** 8 ist ein detaillierteres Blockdiagramm eines Abschnitts des Texturabbildungschips, der den Grenzprüfer **72**, das Cache-Verzeichnis **78**, die Cache-Zugriffsschaltung **82**, den Cache-Speicher **48** und den Texelinterpolator **76** aufweist. Die Texel-Cache-Zugriffseinheit **82** weist vier Cache-Zugriffsbefehl-FIFOs **206A**, **206B**, **206C** und **206D** auf. Die Cache-Zugriffsbefehl-FIFOs **206A** bis **D** speichern Cache-Zugriffsbefehle, die jeweils über die 16-Bit-Busse **84A**, **84B**, **84C** und **84D** von dem Grenzprüfer empfangen werden. Die Cache-Zugriffsbefehl-FIFOs **206A** bis **D** entsprechen jeweils den Steuerungen **200A** bis **200D**, die in **Fig.** 6 gezeigt sind. Beispielsweise rufen die Befehle in dem FIFO **206A** einen Cache-Zugriff des SDRAMs in der Verschachtelung **204A** auf. Bei diesem Ausführungsbeispiel ist jedes Cache-Zugriffsbefehl-FIFO in der Lage, temporär acht 16-Bit-Befehle zu speichern. Um die Pipelineverarbeitungs-Fähigkeit des Systems zu verbessern, können somit acht Befehle in jedem der Cache-Zugriffsbefehl-FIFOs gespeichert werden, bevor die Cache-Zugriffseinheit wirksam wird.

[0105] Wie oben erläutert wurde, vergleicht der Grenzprüfer **72** während einer Aufbereitung die Lese-Cache-Kennung für jeden Texturdatenblock, der auf das Pixel abbildet, auf dem gearbeitet wird, mit jeder der Blockkennungen, die in dem Cache-Verzeichnis **78** gespeichert sind, um zu bestimmen, ob sich das Texel in dem Cache befindet. Wenn ein Treffer auftritt, wird der Blockindex erzeugt, der den Ort des entsprechenden Texturdatenblocks in dem Cache darstellt. Der Kachelbilder/Grenzprüfer implementiert gleichzeitig eine Routine, um die Texeladresse aus den interpolierten Koordinaten S, T, die Textur-ID und die Untertextur-ID des speziellen Texels zu bestimmen, ebenso wie die Tabellennummer der Tabelle, aus der auf das Texel zugegriffen werden soll, und die Größe der Basistabelle der Textur, wie nachfolgend detailliert erklärt wird. Aus dem Blockindex und der Texeladresse (die zusammen die Cache-Adresse bilden) bestimmt dann der Optimierer die spezielle Verschachtelung des Caches, in der das Texel gespeichert ist, sowie die Spalten- und Zeilen-Adreßbits der SDRAM-Chips dieser Verschachtelung, wie oben erklärt wurde. Die Adreßinformationen werden dem entsprechenden Cache-Zugriffsbefehl-FIFO zusammen mit einem Befehl, um den Cache zu lesen, geliefert.

[0106] Der Texelinterpolator **76** weist acht Texeldaten-FIFOs auf, die mit **214A0**, **214A1**, **214B0**, **214B1**, **214C0**, **214C1**, **214D0** und 214D1 bezeichnet sind. Die Texeldaten-FIFOs **214A** und **214A1** sind der Verschachtelung **204A** des Cache-Speichers zugeordnet, die FIFOs **214B0** und **214B1** sind der Verschachtelung **204B** zugeordnet, die FIFOs **214C0** und **214C1** sind der Verschachtelung **204C** zugeordnet und die FIFOs **214D0** und **214D1** sind der Verschachtelung **204D** zugeordnet.

[0107] Wie oben beschrieben wurde, kann durch getrennte Cache-Zugriffswege gleichzeitig auf jede der vier Verschachtelungen des Cache-Speichers zugegriffen werden. Während einer Aufbereitung, wenn die Texel-Cache-Zugriffseinheit **82** auf Texeldaten aus dem Cache-Speicher **48** zugreift, werden über Busse **208A**, **208B**, **208C** und **208D** Texelzugriff-Steuerworte zu dem Cache-Speicher **48** geliefert. Während zweier Rücken-an-Rücken-l6-Bit-Lesezyklen wird gleichzeitig auf vier Texel aus den vier Verschachtelungen zugegriffen. Die vier Texel werden jeweils über Busse **210A**, **210B**, **210C** und **210D** zu einem der Texeldaten-A-FIFOs (**214A0** oder **214A1**), einem der Texeldaten-B-FIFOs (**214B0** oder **214B1**), einem der Texeldaten-C-FIFOs (**214C0** oder **214C1**) und einem der Texeldaten-D-FIFOs (**214D0** oder **214D1**) geliefert. Das Paar von Texeldaten-FIFOs (d.h. 0 und 1), das jeder Verschachtelung A bis D zugeordnet ist, wird in einer abwechselnden Form geladen. Beispielsweise wird ein erstes Texel, das aus der Verschachtelung A gelesen wird, in dem Texeldaten-FIFO **214A0** gespeichert, ein zweites Texel, das aus der Verschachtelung A gelesen wird, wird in dem FIFO **214A1** gespeichert, ein drittes Texel aus der Verschachtelung A wird in dem FIFO **214A0** gespeichert, usw. Dieses abwechselnde Schema wird aus Gründen verwendet, die nachfolgend erläutert werden.

[0108] Jedes der Texeldaten-FIFOs ist zweiunddreißig Bit breit und acht Stufen tief. Zusammen speichern die acht FIFOs **214** acht pipelineartig verarbeitete Stufen, wobei jede Stufe die acht Texel aufweist, die verwendet sind, um resultierende Texeldaten während einer trilinearen Interpolation zu bestimmen. Die Busse **210A**, **210B**, **210C** und **210D** sind sechzehn Bit breit. Jedes SDRAM-Paar in jeder Verschachtelung liefert während jedes Lesezyklusses sechzehn Datenbits. Während jedes Bündels werden die ersten sechzehn Bits von jedem SDRAM-Paar in ein erstes 16-Bit-Register (nicht gezeigt) geliefert, während die nächsten sechzehn Bits von jedem SDRAM-Paar in ein zweites 16-Bit-Register (ebenfalls nicht gezeigt) geliefert werden. Am Ende des zweiten Zyklusses des Bündels werden die Daten aus beiden Registern auf dem entsprechenden 32-Bit-Bus **212A**, **212B**, **212C** oder **212D** geliefert. Um die resultierenden Texeldaten für jedes Pixel zu bestimmen, greift der Texelinterpolator **76** auf die FIFOs zu, um die nächste Stufe der acht Texel zu lesen, und interpoliert diese Texel auf die Art und Weise, die oben beschrieben ist. Die resultierenden Texeldaten werden dann über den Bus **28** zu der Rahmenpufferplatine **14** (**Fig.** 2) geliefert, wo dieselben auf die oben beschriebene Art und Weise bei der Aufbereitung des Anzeigebildschirmpixels verwendet werden.

[0109] Wenn eine trilineare Interpolation durchgeführt wird, werden die resultierenden Texeldaten für jedes Pixel aus vier Texeln in einer MIP-Tabelle und vier Texeln in einer benachbarten MIP-Tabelle interpoliert. Benachbarte Anzeigebildschirmpixel werden im allgemeinen aufeinanderfolgend aufbereitet. Häufig werden benachbarte Anzeigebildschirmpixel auf benachbarte Orte in einer Textur-MIP-Tabelle abbilden. Folglich ist es üblich, daß einige gemeinsame Texeldaten beim Interpolieren der resultierenden Texeldaten für nacheinander aufbereitete Grundelemente verwendet werden können. Bei einem Ausführungsbeispiel der Erfindung wird nur bei dem ersten Lesen auf den Cache zugegriffen, wobei die Cache-Lesezyklen für jedes nachfolgende Lesen gespeichert werden, wenn innerhalb einer Anzahl von wenig beabstandeten Lesezyklen mehrere Male auf gemeinsame Texeldaten zugegriffen wird. Die am kürzesten vorher gelesenen Texel werden in den Texeldaten-FIFOs gespeichert. Somit werden nachfolgende Zugriffe auf diese Texel aus den FIFOs und nicht aus dem Cache durchgeführt. Dies reduziert die erforderliche Anzahl von Cache-Zugriffen, wodurch die Systembandbreite erhöht wird.

[0110] Wenn die Texeldaten, die zuletzt für ein vorheriges Pixel in einen der Texeldaten-FIFOs 0 oder 1 geschrieben wurden, mit den Texeldaten für ein Pixel, das gegenwärtig in der Pipelineposition für einen Zugriff auf den Cache ist, übereinstimmen, wird für jeden der Texeldatenwege A, B, C und D kein Cache-Zugriffsbefehl zu dem entsprechenden Cache-Zugriff-FIFO **206A**, **B**, **C** oder **D** geliefert. Stattdessen wird ein Befehl zu dem Texelinterpolator gesendet, um anzuzeigen, daß die Texeldaten in dem zuletzt beschriebenen Ort des entsprechenden Texeldaten-FIFOs **214A**, **B**, **C**, oder **D** sind. Für jeden der Wege A, B, C und D, bei dem die Texeldaten, die dem Pixel entsprechen, das gegenwärtig an der Pipelineposition zum Zugreifen auf den Cache ist, nicht mit den Daten in dem zuletzt beschriebenen Ort des entsprechenden Texeldaten-FIFOs entsprechen, wird ein Texel-Cache-Zugriffsbefehl zu dem entsprechenden Texel-Cache-Zugriffsbefehl-FIFO geliefert, um diese Texeldaten aus dem Cache-Speicher **48** zu lesen.

[0111] Es sollte offensichtlich sein, daß für einige der Verschachtelungen A bis D für ein beliebiges Pixel, das sich gegenwärtig an der Pipelineposition befindet, für das ein Cache-Zugriff betrachtet werden muß, ein unterschiedliches Ergebnis auftreten kann. Beispielsweise können gemeinsame Texeldaten für aufeinanderfolgende Pixel für die Verschachtelung A existieren, jedoch nicht für die Verschachtelungen B bis D. In einem solchen Fall werden die Texeldaten für das zweite und die nachfolgenden Pixel an der Pipelineposition zum Zugreifen auf Texeldaten aus dem Cache aus den Verschachtelungen B bis D gelesen, während die Texeldaten von der Verschachtelung A für dieses zweite Pixel aus dem gleichen Ort von einem der Texeldaten-FIFOs **214A0** oder **214A1** gelesen wird. Das vorliegende Schema liefert Bandbreiten-Einsparungen, wenn Texel für mehrere Pixel aus den Texeldaten-FIFOs neu gelesen werden, ohne auf den Cache zuzugreifen.

[0112] Der Texelinterpolator **76** weist ein Texelinterpolator-Befehls-FIFO **216** auf, das 53-Bit-Befehle von dem Grenzprüfer **72** über einen 53-Bit-Bus **218** empfängt. Das Texelinterpolator-Befehls-FIFO kann bis zu sechzehn Befehle speichern, die dem Interpolator anzeigen, welche Texeldaten-FIFO-Orte die Texeldaten enthalten, die während jedes Zyklusses beim Interpolieren der resultierenden Texeldaten verwendet werden sollen. Die Interpolatorbefehle zeigen ferner den Interpolationsmodus (d.h. Punktabtastung, bilinear oder trilinear) an, und weisen die Gradienten und Bruch-Werte der Koordinaten S und T auf, die die Art und Weise spezifizieren, auf die jedes Texel bei der Interpolation gewichtet werden soll. Die Befehle weisen Daten auf, die anzeigen, von welchen Texeldaten-FIFOs **214A0**, **A1**, **B0**, **B1**, **C0**, **C1**, **D0** oder **D1** jedes der vier (bilinear) oder acht (trilinear) Texel gelesen werden soll, und ob die Texeldaten neu oder alt sind. Texeldaten sind neu, wenn sie sich von den Texeldaten unterscheiden, die in dem zuletzt beschriebenen Ort eines Texeldaten-FIFOs dieses Wegs gespeichert sind. Wenn sie neu sind, ist ein Cache-Lesen erforderlich. Texeldaten sind alt, wenn sie die gleichen sind, wie die, die in dem zuletzt beschriebenen Ort eines Texeldaten-FIFOs gespeichert sind. Wenn sie alt sind, ist ein Cache-Lesen nicht erforderlich. Wenn die Texeldaten neu sind, muß der FIFO-Lesezeiger auf einen nächsten Ort in dem FIFO bewegt werden, wohingegen, wenn die Texeldaten alt sind, die gleichen Daten aus dem gleichen FIFO-Ort gelesen werden, und der Lesezeiger nicht bewegt werden muß.

[0113] Das folgende Beispiel, das bezugnehmend auf die **Fig.** 9 und 10 erklärt wird, zeigt den Betrieb der Texelzugriffsschaltung, die in **Fig.** 8 gezeigt ist, genauer. **Fig.** 9 zeigt mehrere Texel einer oberen MIP-Tabelle und mehrere Texel einer unteren (kleineren) MIP-Tabelle. Die Texel sind mit An, Bn, Cn und Dn bezeichnet (wobei n eine ganze Zahl darstellt), gemäß dem Bezeichnungsschema, das vorher bezugnehmend auf **Fig.** 7 beschrieben wurde. Sieben Pixel, die aufbereitet werden sollen, sind mit P0, P1,... P6 bezeichnet. Wie gezeigt ist, bilden die Pixel, die aufbereitet werden sollen, nicht direkt auf die Texel der MIP-Tabellen ab. Bei diesem Beispiel wird eine trilineare Interpolation durchgeführt, derart, daß auf vier Texel aus der oberen Tabelle und auf vier Texel aus der unteren Tabelle zugegriffen werden muß und daß dieselben für jedes Pixel interpoliert werden. Die Schrittrichtung ist die Richtung der Aufbereitung und entspricht der numerischen Numerierung der Pixel.

[0114] **Fig.** 10 zeigt den Cache-Zugriffsbefehl-FIFO (**206A**), den Texeldaten-FIFO A0 (**214A0**), den Texeldaten-FIFO A1 (**214A1**) und den Texelinterpolatorbefehl-FIFO **216**. Nur die FIFOs, die dem Texeldatenweg A zugeordnet sind, sind der Bequemlichkeit halber gezeigt, da die FIFOs für jeden anderen der Texeldatenwege B, C und D auf die gleiche Art und Weise arbeiten. Jeder FIFO-Puffer weist einen Schreibzeiger und einen Lesezeiger auf, die jeweils auf einzelne Orte in dem FIFO zeigen, an die Daten geschrieben werden sollen oder von denen Daten gelesen werden sollen. Die Zeiger können bei diesem darstellenden Ausführungsbeispiel um einen Ort einzeln bewegen.

[0115] Das Pixel P0 bildet auf die Texel A0, B0, C0 und D0 in der oberen Tabelle und die Texel A0, B0, C0 und D0 in der unteren Tabelle ab, so daß diese acht Texel interpoliert werden, um die resultierenden Texeldaten für das Pixel P0 zu erzeugen. Für das Pixel P0 wird die Adresse des Texels A0 in der oberen Tabelle (d.h. uA0) in einen ersten Ort in dem Cache-Zugriffsbefehl-FIFO **206A** geschrieben, zusammen mit einer Adresse, die anzeigt, daß das Texeldaten-FIFO **214A0** mit den Texeldaten beschrieben werden soll, die an dieser Adresse aus dem Cache gelesen werden. Als nächstes wird der Schreibzeiger des Cache-Zugriffbefehl-FIFO **206A** um einen Ort bewegt, und die Adresse des Texels A0 in der unteren Tabelle (d.h. lA0) wird in diesen nächsten FIFO-Ort geschrieben, zusammen mit einer Adresse, die anzeigt, daß das Texeldaten-FIFO **214A1** mit den Texeldaten beschrieben werden soll, die an dieser Adresse aus dem Cache gelesen werden. Auf diese Art und Weise werden die Texeldaten-FIFOs null und eins aus den oben erläuterten Gründen abgewechselt. Die Cache-Zugriffsbefehl-FIFOs **206B** bis **D** werden auf eine gleichartige Art und Weise, die sich auf die Texel B0, C0 und D0 in der oberen und unteren Tabelle bezieht, aktualisiert.

[0116] Für das Pixel P1 müssen die Texel A1 in der oberen und der unteren Tabelle, die jeweils an den Adressen uA1 und lA1 gespeichert sind, interpoliert werden. Da die Texel A1 in der oberen und der unteren Tabelle neue Texel sind und nicht den Texeln von dem vorherigen Pixel P0 entsprechen, wird aus dem Cache auf dieselben zugegriffen. Folglich werden die Texeladressen für diese Texel den nächsten zwei Orten des Cache-Zugriffbefehl-FIFO **206A** hinzugefügt, zusammen mit den entsprechenden Adressen, die jeweils anzeigen, daß die Texeldaten, die aus diesen Adressen gelesen werden, in den Texeldaten-FIFOs **214A0** und **214A1** gespeichert werden müssen. **Fig.** 10 stellt den Cache-Zugriffbefehl-FIFO **206A** dar, nachdem derselbe mit diesen Informationen aktualisiert wurde.

[0117] Da für die ersten zwei Pixel P0 und P1 keine gemeinsamen adressierten Texel A existieren, wird auf den Cache-Speicher zugegriffen, um die Texeldaten für beide wiederzugewinnen. Der erste Befehl wird aus dem Cache-Zugriffbefehl-FIFO **206A** gelesen, was bewirkt, daß die Texeldaten an der Adresse uA0 aus dem Cache-Speicher gelesen werden und in den ersten Ort des Texeldaten-FIFOs **214A0** geschrieben werden. Dann wird der nächste Befehl aus dem Cache-Zugriffbefehl-FIFO gelesen, auf die Texeldaten an der Adresse lA0 aus dem Cache zugegriffen und dieselben an den ersten Ort des Texeldaten-FIFOs **214A1** geschrieben. Danach wird der nächste Befehl aus dem Cache-Zugriffbefehl-FIFO gelesen, auf die Texeldaten an der Adresse uA1 aus dem Cache zugegriffen und dieselben an den nächsten Ort in dem Texeldaten-FIFO **214A0** geschrieben. Schließlich wird der vierte Befehl aus dem Cache-Zugriffbefehl-FIFO gelesen, auf die Texeldaten an der Adresse lA1 aus dem Cache zugegriffen und dieselben werden an den nächsten Ort des Texeldaten-FIFOs **214A1** geschrieben.

[0118] Für das nächste Pixel P2, das aufbereitet werden soll, müssen die Texel an den Adressen uA1 und lA1 interpoliert werden. Da für das vorher aufbereitete Pixel P1 auf dieselben zugegriffen wurde, sind sie jeweils in den zuletzt geschriebenen Einträgen in den Texeldaten-FIFOs **214A0** und **214A1** gespeichert. Folglich werden keine neuen Cache-Zugriffbefehle für diese Texel zu dem Cache-Zugriffbefehl-FIFO **206A** geliefert. Vielmehr kann, nachdem die resultierenden Texeldaten für das Pixel P1 interpoliert sind, auf die Texeldaten, die an den Adressen uA1 und lA1 gespeichert sind, jeweils durch den Texelinterpolator aus den zuletzt gelesenen Orten der Texeldaten-FIFOs **214A0** und **214A1** zugegriffen werden, ohne auf den Cache zugreifen zu müssen. Das Lesen von Daten direkt aus einem FIFO-Puffer verbraucht weniger Zeit als das Zugreifen auf Daten aus einem Cache-Speicher. Daher erhöhen die FIFO-Puffer der vorliegenden Erfindung, die Cache-Zugriffe reduzieren, die Systembandbreite.

[0119] Wie oben erläutert wurde schließen die Texeldaten-FIFOs **214**, die jeder der Verschachtelungen A bis D zugeordnet sind, getrennt gesteuerte FIFOs null und eins ein. Die FIFOs sind auf diese Weise unterteilt, um effizient eine trilineare Interpolation zu implementieren. Wie aus dem vorhergehenden offensichtlich sein sollte, liefert bei dem oben beschriebenen Ausführungsbeispiel jeder Texeldaten-FIFO **214** einen Zugriff auf seinen zuletzt gelesenen Eintrag, indem sein Lesezeiger belassen wird, um für aufeinanderfolgende Lesevorgänge auf den gleichen Eintrag zu zeigen. Obwohl jede Verschachtelung während aufeinanderfolgender Lesezyklen zwischen Lesevorgängen von zwei Tabellen abwechselt, können die getrennten FIFOs folglich aufeinanderfolgende Lesevorgänge in einer einzelnen Tabelle durchführen, was ermöglicht, daß der Lesezeiger bei aufeinanderfolgenden Zugriffen auf den FIFO auf die

gleichen Texeldaten zeigt.

[0120] Während jedes Pixel durch den Kachelbilder/Grenzprüfer **72** bearbeitet wird und Befehle zu dem Cache-Zugriffbefehl-FIFO geliefert werden, werden auch Befehle in den Texelinterpolatorbefehl-FIFO **216** geschrieben. Wenn beispielsweise der Befehl, auf das Texel an der Adresse uA0 zuzugreifen, zu dem Cache-Zugriffbefehl-FIFO für das Pixel P0 geliefert wird, wird der Befehl Neu0 zu dem ersten Ort des Texelinterpolatorbefehl-FIFOs **216** geliefert. Der Befehl Neu0 zeigt dem Texelinterpolator an, daß auf die nächsten Texeldaten von der Verschachtelung A aus dem Cache zugegriffen wird und dieselben zu dem Texeldaten-FIFO **214A0** geliefert werden, was anzeigt, daß, um die Texeldaten aus dem FIFO zu lesen, der Texelinterpolator den FIFO-Lesezeiger um einen Ort von dem Ort, der zuletzt gelesen wurde, bewegen soll.

[0121] Für den nächsten Befehl, der zu dem Cache-Zugriffbefehl-FIFO geliefert wird, der der Texeladresse lA0 entspricht, wird der Befehl Neu1 zu dem nächsten Ort des Texelinterpolatorbefehl-FIFOs geliefert. Der Befehl Neu1 zeigt dem Texelinterpolator an, daß die nächsten Texeldaten von der Verschachtelung A ebenfalls neu sind und aus dem Texeldaten-FIFO **214A1** gelesen werden sollen. In gleicher Weise werden für die Befehle, die den Texeladressen uA1 und lA1, die dem Pixel P1 entsprechen, zugeordnet sind, jeweils die Befehle Neu0 und Neu1 an die nächsten zwei Orte des Texelinterpolatorbefehl-FIFOs **216** geschrieben.

[0122] Da die Texeldaten an den Adressen uA1 und lA1 identisch zu den Daten sind, die für das vorherige Pixel P1 in die FIFOs geschrieben wurden, sind für das Pixel P2 die Befehle, die an die nächsten zwei Orte des Texelinterpolatorbefehl-FIFOs **216** geschrieben werden, Alt0 und Alt1, die dem Texelinterpolator jeweils anzeigen, daß die nächsten Texeldaten wieder aus den zuletzt gelesenen Orten der Texeldaten-FIFOs **214A0** und **214A1** gelesen werden sollen. Die Befehle Alt0 und Alt1 zeigen an, daß, um die nächsten Texeldaten aus den FIFOs zu lesen, der Texelinterpolator den FIFO-Lesezeiger nicht von dem Ort, der zuletzt gelesen wurde, bewegen soll.

[0123] In **Fig.** 9 sind drei Tabellen dargestellt: die erste Tabelle zeigt die Texel, die für jedes der Pixel interpoliert werden müssen, die zweite Tabelle gibt die getrennten Texeldatenwerte an, die in den Texeldaten-FIFOs A0, B0, C0 und D0 gespeichert werden müssen; und die dritte Tabelle gibt die getrennten Texeldatenwerte an, die in den Texeldaten-FIFOs A1, B1, C1 und D1 gespeichert werden müssen. Die Freiräume zeigen gemeinsam verwendete Texeldaten an, die vorher aus dem Cache gelesen wurden und nicht wiederum aus dem Cache gelesen werden müssen, wobei stattdessen aus den FIFOs auf dieselben zugegriffen werden kann. Wenn resultierende Texeldaten für mehrere Pixel interpoliert werden, kann durch das FIFO-Schema der vorliegenden Erfindung eine große Anzahl von Cache-Zugriffen eingespart werden, wie dieses Diagramm zeigt, was eine Zunahme der Systembandbreite zur Folge hat.

[0124] **Fig.** 11 ist ein Blockdiagramm einer Schaltung, die durch den Texturabbildungschip verwendet wird, um zu bestimmen, ob bei jeder Verschachtelung Texeldaten, die für ein Pixel gelesen werden sollen, für das zuletzt aufbereitete Pixel gelesen wurden. Diese Schaltung wird verwendet, um zu bestimmen, ob ein neuer Befehl in einen der Cache-Zugriffbefehl-FIFOs geschrieben werden soll, um zu bewirken, daß neue Daten aus dem Cache gelesen werden, oder ein Befehl in den Texelinterpolatorbefehl-FIFO geschrieben werden soll, der anzeigt, daß die Texeldaten alt sind und aus einem der Texeldaten-FIFOs gelesen werden sollen. **Fig.** 11 zeigt nur eine einzelne Schaltung, die der Verschachtelung A zugeordnet ist. Jedoch sind gleichartige Schaltungen für die Verschachtelungen B, C und D vorgesehen. Die Schaltung befindet sich in dem Optimiererelement des Kachelbilders/Grenzprüfers. Aus dem interpolierten S-, T-Wert, der für jedes Texel, das interpoliert werden soll, durch den Kachelbilder/Grenzprüfer empfangen wird, liefert der Optimierer eine Texeladresse (einschließlich der Blockkennung und der Texeladresse) auf einem Bus **220A**. Die Adressen der zuletzt verarbeiteten Texel, die den Texeldaten-FIFOs **214A0** und **214A1** zugewiesen sind, sind jeweils in Adreßregistern **222A0** und **222A1** gespeichert. Die momentane Texeladresse wird jeweils durch Komparatoren **224A0** und **224A1** mit den Texeladressen, die in den Registern **222A0** und **222A1** gespeichert sind, verglichen.

[0125] Wenn die momentane Texeladresse nicht mit einer der Adressen, die in den Registern **222A0** und **222A1** gespeichert sind, übereinstimmt, muß auf die Texeldaten, die dieser Texeladresse entsprechen, aus dem Cache-Speicher zugegriffen werden, wobei der entsprechende Befehl in den Cache-Zugriffbefehl-FIFO geschrieben wird. Wenn jedoch die Texeladresse mit der Adresse, die in dem Adreßregister **222A0** oder **222A1** gespeichert ist, übereinstimmt, werden die Texeldaten jeweils in dem Texeldaten-FIFO **212A0** oder **212A1** gespeichert, an dem Ort, der von dem Texelinterpolator gelesen wird, unmittelbar bevor auf die Texeldaten zugegriffen wird, die der Adresse entsprechen. Daher wird kein Cache-Zugriffbefehl in den Cache-Zugriffbefehl-FIFO geschrieben, während ein Befehl in den entsprechenden Texelinterpolatorbefehl-FIFO geschrieben wird, der anzeigt, daß die Texeldaten alt sind, und daß aus dem zuletzt gelesenen FIFO-Ort auf dieselben zugegriffen werden soll, ohne den Lesezeiger zu bewegen.

V. Organisation der Texturdatenblöcke

[0126] **Fig.** 1 zeigt eine Reihe von quadratischen Textur-MIP-Tabellen, die eine Basistabelle **100** von 8 × 8 Texeln einschließen. Aus der Basistabelle ist jede nachfolgende Tabelle größenmäßig gefiltert, bis zu einer kleinsten Tabelle 108 (die nur ein Texel aufweist). Der kleinsten Tabelle 108 ist eine Tabellennummer von null zugewiesen, wobei die Tabellennummer für jede sukzessive größere Tabelle um eins inkrementiert ist, so daß die Basistabelle **100** bei diesem Beispiel eine Tabellennummer von drei besitzt. Die Tabellennummer wird beim Bestimmen der Blockkennung für jeden Texturdatenblock verwendet, auf eine Art und Weise, die nachfolgend beschrieben wird. Gemäß diesem Tabellennumerierungsschema entspricht unter der Annahme einer quadratischen Texturbasistabelle eine Tabellennummer von zehn einer Tabelle von 1024 × 1024 Texeln, eine Tabellennummer von neun stellt eine Tabelle mit 512 × 512 Texeln dar, eine Tabellennummer von acht stellt eine Tabelle mit 256 × 256 Texeln dar, usw. Wenn die Texturbasistabelle nicht quadratisch ist, entspricht eine Tabellennummer von zehn einer Tabelle, deren größere Dimension 1024 Texel ist. Obwohl bei dieser Erläuterung eine quadratische Texturbasistabelle angenommen ist, sind auch rechteckige Tabellen möglich. Bei rechteckigen Tabellen ist die Tabellennummer durch die Anzahl von Texeln der längeren Dimension der Tabelle bestimmt. Beispielsweise weist eine rechteckige Tabelle mit einer Tabellennummer von zehn eine längere Dimension mit 1024 Texeln auf. Es sollte offensichtlich sein, daß alternativ andere Tabellennumerierungsschemas verwendet werden können.

[0127] Eine quadratische Tabelle mit 1024 × 1024 Texeln, die eine Tabellennummer von zehn aufweist, erfordert zehn Bit von S-Koordinaten S[9:0] und zehn Bit von T-Koordinaten T[9:0], um den Ort jedes Texels in der Tabelle eindeutig zu identifizieren. In gleicher Weise erfordert eine Tabelle, die eine Tabellennummer von neun aufweist, neun Bit von sowohl S- als auch T-Koordinaten, um den Ort jedes Texels zu identifizieren, eine Tabelle mit einer Tabellennummer von acht benötigt acht Bit von sowohl S- als auch T-Koordinaten, um den Ort jedes Texels zu identifizieren, usw. Die Koordinaten S und T, die eindeutig den Ort eines Texels in einer MIP-Tabelle identifizieren, die einem beliebigen Pixel entsprechen, werden auf die oben beschriebene Art und Weise interpoliert.

[0128] Wie nachfolgend detaillierter beschrieben wird, sind Texturdaten in Blöcken von 256 × 256 Texeln in dem Hauptspeicher **17** des Hostcomputers **15** (**Fig.** 2) gespeichert. Wenn ein Cache-Fehlzugriff auftritt, wird eine Lese-Cache-Kennung, die den Texturdatenblock identifiziert, der den Fehlzugriff in dem Cache verursachte, durch den Hostcomputer gelesen, wobei dieser Texturdatenblock dann in den Cache-Speicher **48** der Texturabbildungsplatine heruntergeladen wird. Bei dem beschriebenen darstellenden Ausführungsbeispiel der Erfindung können jeweils vierundsechzig Texturdatenblöcke in dem Cache-Speicher gespeichert sein. Diese vierundsechzig Texturdatenblöcke können Daten von mehreren MIP-Tabellen von einer oder mehreren Texturen aufweisen. Jeder Block weist eine zugeordnete Blockkennung auf, die denselben eindeutig identifiziert. MIP-Tabellen mit einer Tabellennummer von neun oder größer weisen mehr als 256 × 256 Texel auf, und sind daher in mehreren Blöcken gespeichert. Die S-, T-Koordinaten hoher Ordnung für jede Tabelle, die in mehreren Blöcken gespeichert ist, befinden sich in der Blockkennung für die Datenblöcke, die die Tabelle speichern.

[0129] Beispielsweise weisen MIP-Tabellen mit einer Tabellennummer von neun eine Dimension auf, die gleich 512 Texel ist, und sind, wenn quadratisch, 512 × 512 Texel groß. Die Tabelle ist in vier Blöcke von 256 × 256 Texel geteilt (unter der Annahme einer quadratischen Texturtabelle). Daher weist die Blockkennung für jeden dieser Blöcke ein S-Koordinatenbit hoher Ordnung und ein T-Koordinatenbit hoher Ordnung auf (d.h. S[8] und T[8]), die den Ort des Blocks in der Tabelle identifizieren. In gleicher Weise sind MIP-Tabellen mit einer Tabellennummer von zehn 1024 × 1024 Texel groß und in sechzehn Datenblöcke unterteilt. Daher weisen die Blockkennungen für jeden dieser Blöcke zwei S-Koordinatenbits hoher Ordnung und zwei T-Koordinatenbits hoher Ordnung auf (d.h. S[9:8] und T[9:8]), die den Ort des Blocks in der Tabelle identifizieren.

[0130] Wie nachfolgend beschrieben wird, sind die Textur-MIP-Tabellen unterteilt und in einem Speicher gespeichert, so daß die gleichen Abschnitte benachbarter MIP-Tabellen in gegenüberliegenden SDRAM-Bänken gespeichert sind, um die Systembandbreite während einer trilinearen Interpolation zu reduzieren. Außerdem können mehrere Tabellen, die kleiner als 256 × 256 Texel sind, in einem einzelnen Block des Cache-Speichers gespeichert sein, um eine effiziente Verwendung des Speicherraums in dem Cache-Speicher zu liefern.

[0131] **Fig.** 12 zeigt einen Satz von Textur-MIP-Tabellen für eine spezielle Textur, die folgendes Oberflächenbild aufweist:
LA
95

[0132] Wie in **Fig.** 12 gezeigt ist, ist jede MIP-Tabelle in der Reihe von MIP-Tabellen für eine Textur in vier Quadranten unterteilt, die für eine quadratische Texturtabelle eine gleiche Größe aufweisen. Bei dem Beispiel, das in **Fig.** 12 gezeigt ist, hat die Basistabelle eine Tabellennummer von neun und ist in Quadranten **9Q1** (der das Bild L einschließt), **9Q2** (der das

Bild A einschließt), **9Q3** (der das Bild 9 einschließt), und **9Q4** (der das Bild 5 einschließt) unterteilt. In gleicher Weise ist die Tabellen mit der Nummer acht in Quadranten **8Q1**, **8Q2**, **8Q3**, **8Q4** unterteilt, die die Bilder L, A, 9 bzw. 5 einschließen. In gleicher Weise ist die Tabelle mit der Nummer sieben in Quadranten **7Q1**, **7Q2**, **7Q3**, **7Q4** unterteilt, die die Bilder L, A, 9 bzw. 5 einschließen. Die kleineren Tabellen sind in gleicher Weise in Quadranten unterteilt.

[0133] Zwei Quadranten jeder MIP-Tabelle sind in einer Bank der SDRAMs gespeichert, die den Cache bilden, während die anderen zwei Quadranten in der gegenüberliegenden Bank gespeichert sind. Gemäß dem Texturdaten-Zuweisungsschema der Erfindung sind für Texturen, die eine Basistabelle mit einer Nummer größer oder gleich acht aufweisen (die eine Größe größer oder gleich 256 × 256 Texel aufweisen), die Speicherorte in den Blöcken des Speicherraums für alle Quadranten aller MIP-Tabellen dieser Textur vordefiniert. Beispielsweise sind die Quadranten **9Q1** und **9Q4** der Tabellennummer neun in getrennten Blöcken in der Cache-Bank eins gespeichert, während die Quadranten **9Q2** und **9Q3** in getrennten Blöcken der Cache-Bank null gespeichert sind, wie in **Fig.** 13 gezeigt ist. Die entsprechenden Quadranten der benachbarten MIP-Tabellen sind in Blöcken in gegenüberliegenden Bänken gespeichert. Folglich sind bei diesem Beispiel die Quadranten **8Q1** und **8Q4**, die jeweils die Block-gefilterten Texturdaten der Quadranten **9Q1** und **9Q4** aufweisen, in dem gleichen Block in der Cache-Bank null gespeichert. In gleicher Weise sind die Quadranten **8Q2** und **8Q3**, die jeweils die Block-gefilterten Texturdaten der Quadranten **9Q2** und **9Q3** aufweisen, in dem gleichen Block in der Cache-Bank eins gespeichert. **Fig.** 13 ist nicht maßstabsgerecht bezüglich **Fig.** 12. Es sollte offensichtlich sein, daß die Tabellenquadranten von **Fig.** 12 die gleiche Größe aufweisen wie die von **Fig.** 13, da dieselben identisch sind.

[0134] Aufgrund der jeweiligen Größen der Tabellen besetzt jeder Quadrant einer Tabelle mit der Nummer neun einen kompletten Block von 256 × 256 Texeln, wohingegen die Quadranten einer Tabelle mit der Nummer acht jeweils nur ein Viertel eines Blocks besetzen. Daher besetzen die Quadranten **8Q2** und **8Q3** zusammen die Hälfte des gleichen Blocks und die Quadranten **8Q1** und **8Q4** besetzen die Hälfte eines weiteren Blocks in der gegenüberliegenden Bank. Um den Cache-Speicherraum effizient zuzuweisen, sind die unbesetzten Orte in jedem dieser Blöcke durch entsprechende Quadranten von Tabellen besetzt, die eine Tabellennummer von sieben oder weniger aufweisen. Daher besetzen alle Tabellen, die Nummern null bis acht aufweisen, zusammen zwei Blöcke, wobei jeder der zwei Blöcke in einer getrennten Bank ist. Die Orte der Quadranten für die Tabellen, die Tabellennummern von acht oder kleiner aufweisen (unter der Voraussetzung einer Basistabelle mit einer Tabellennummer von acht oder größer) sind auf die Art und Weise vordefiniert, die in **Fig.** 13 gezeigt ist. Wie gezeigt ist, bewahren der obere rechte Quadrant **8Q2** und der untere linke Quadrant **8Q3** die gleiche physikalische Beziehung und besetzen jeweils den oberen rechten und den unteren linken Quadranten eines ersten Blocks, während der obere linke Quadrant **8Q1** und der untere rechte Quadrant **8Q4** ebenfalls die gleiche physikalische Beziehung bewahren und jeweils den oberen linken und den unteren rechten Quadranten eines zweiten Blocks besetzen, der sich in einer von dem ersten Block unterschiedlichen Bank befindet. Ferner bewahren die Quadranten **7Q1** und **7Q4** die gleiche physikalische Beziehung und besetzen den oberen linken Quadranten des ersten Blocks, während die Quadranten **7Q2** und **7Q3** die gleiche physikalische Beziehung bewahren und jeweils den oberen rechten Quadranten des zweiten Blocks besetzen.

[0135] Wenn ein Pixel auf eine Position in der Texturtabelle abbildet, die zwischen vier Texeln in einer MIP-Tabelle und vier Texeln in einer benachbarten MIP-Tabelle liegt, dann wird während einer trilinearen Interpolation auf alle acht Texel aus dem Cache zugegriffen. Die Texel, auf die aus beiden MIP-Tabellen zugegriffen wird, weisen gemeinsame Texturdaten auf, wobei die Daten aus der kleineren Tabelle eine gefilterte Version der Daten aus der größeren Tabelle sind. Wie oben erläutert wurde, bilden, wenn Pixel eines Objekts aufbereitet werden, benachbarte Pixel häufig auf die gleichen zwei MIP-Tabellen für die Textur ab, was erfordert, daß Lesevorgänge aus dem Cache kontinuierlich zwischen den Cache-Blöcken, die die zwei Tabellen speichern, schalten. Durch das Speichern gemeinsamer Daten aus benachbarten MIP-Tabellen in unterschiedlichen Bänken der Cache-SDRAM-Chips, werden keine Seitenumladungsnachteile erlitten, wenn Cache-Lesevorgänge während aufeinanderfolgenden Lesezyklen zwischen zwei MIP-Tabellen schalten. Dies liefert eine effiziente Implementierung einer trilinearen Interpolation.

[0136] Aus dem vorhergehenden ist offensichtlich, daß, wenn eine Textur eine Basistabelle mit einer Tabellennummer von acht oder größer aufweist, die Zuweisung der MIP-Tabellen unter den Blöcken für diese Textur gemäß dem beschriebenen veranschaulichenden Ausführungsbeispiel der Erfindung vordefiniert ist. Der Grund dafür ist, daß zwei Quadranten einer Tabelle mit einer Tabellennummer acht bestimmte vordefinierte Orte eines ersten Blocks in einer der Bänke besetzen, während die anderen zwei Quadranten der Tabelle mit einer Tabellennummer acht bestimmte gegenüberliegende, vordefinierte Orte in einem anderen Block der gegenüberliegenden Bank besetzen, wie oben erläutert wurde und in **Fig.** 13 gezeigt ist. Jedoch sind für Texturen mit einer Basistabelle mit einer Tabellennummer von sieben oder kleiner mehrere Orte in den zwei Speicherblöcken (ei-

nem Block in jeder Bank) verfügbar, um die Tabellen zu speichern, und werden durch den Hostcomputer ausgewählt. Wenn Abschnitte der mehreren Tabellen einen einzelnen Datenblock gemeinsam verwenden, wird eine Untertextur-Identifikation (ID) zugewiesen, auf eine Art und Weise, die nachfolgend beschrieben wird, um den Ort jeder Tabelle in dem gemeinsam verwendeten Block zu identifizieren.

[0137] Zusätzlich zu der Organisation der Reihe von MIP-Tabellen von **Fig.** 12, zeigt **Fig.** 13 ferner die Art und Weise, auf die eine zweite Reihe von MIP-Tabellen von einer unterschiedlichen Textur (d.h. ein Schachbrettmuster) unter den Speicherblöcken zugewiesen ist. Die MIP-Tabellen dieser zweiten Textur sind auf die gleiche Art und Weise wie die erste Textur unterteilt und in getrennten Datenblöcken gespeichert. Obwohl die Organisation von **Fig.** 13 zeigt, daß die MIP-Tabellen unterschiedlicher Texturen in getrennten Blöcken organisiert sind, sollte es offensichtlich sein, daß Texturdaten von zwei unterschiedlichen Texturen in dem gleichen Block gespeichert sein können.

[0138] Wie oben erläutert wurde, kann der Cache-Speicher bei einem veranschaulichenden Ausführungsbeispiel bis zu vierundsechzig Blöcke von Texturabbildungsdaten speichern, wobei jeder Block 256 × 256 Texel aufweist. Der Cache-Speicher ist in zwei Bänke geteilt, wobei die Blöcke 0 bis 31 in der Bank null liegen und die Blöcke 32 bis 63 in der Bank eins liegen. Das Cache-Verzeichnis weist bis zu vierundsechzig Blockkennungseinträge auf, die den Blöcken in dem Cache zugeordnet sind. Der physikalische Ort jeder Blockkennung in dem Cache-Verzeichnis identifiziert den physikalischen Ort des entsprechenden Texturdatenblocks in dem Cache-Speicher. Aus der Blockkennung wird ein Blockindex erzeugt, der den Ort des Blocks anzeigt. Die Cache-Adresse für jedes Texel in dem Cache wird aus dem Blockindex für den Block und der Texeladresse in dem Block gebildet. Die Texeladresse weist die interpolierten S-, T-Koordinaten niedriger Ordnung für das Texel auf und kann ferner Bits von der Untertextur-ID aufweisen, wie nachfolgend erläutert wird.

[0139] **Fig.** 14 zeigt ein Beispiel einer Textur-MIP-Tabelle mit einer Tabellennummer von neun, die in vier Quadranten unterteilt ist. Die MIP-Tabelle ist 512 × 512 Texel groß, weshalb jeder Quadrant 256 × 256 Texel groß ist und einem einzelnen Speicherblock entspricht. Gemäß einem Ausführungsbeispiel der vorliegenden Erfindung ist durch den Hostcomputer ein einfaches Schema implementiert, um die Bank in dem Cache zu bestimmen, der jeder Quadrant der MIP-Tabelle zugewiesen werden soll. Wie nachfolgend erklärt wird, diktieren für jeden MIP-Tabellenquadranten die Ergebnisse einer logischen Exklusiv-ODER-Operation auf den Werten der höchstwertigen Bits der Koordinaten S und T für den Quadranten die SDRAM-Bank in dem Cache, der der Quadrant zugewiesen ist.

[0140] Für eine Tabelle von 512 × 512 Texeln spezifizieren neun S-Koordinatenbits S[8:0] und neun T-Koordinatenbits T[8:0] den Ort jedes Texels in der Tabelle. Die Quadrantengrenzen sind an dem Mittelpunkt der Tabelle sowohl in die S- als auch die T-Dimension eingerichtet, die durch die höchstwertigen S- und T-Koordinatenbits S[8] und T[8] dargestellt sind. Um die Cache-Bänke für jeden der vier Quadranten einer MIP-Tabelle mit einer Tabellennummer von neun zu bestimmen, wird daher eine Exklusiv-ODER-Operation für jeden Quadranten auf den Werten seiner entsprechenden höchstwertigen S- und T-Koordinatenbits S[8] und T[8] durchgeführt. In gleicher Weise wird für eine MIP-Tabelle mit einer Tabellennummer von zehn die Cache-Bank für jeden Quadranten durch eine Exklusiv-ODER-Operation auf den entsprechenden Werten seiner höchstwertigen S- und T-Koordinatenbits S[9] und T[9] bestimmt. Für MIP-Tabellen mit einer ungeradzahligen Tabellennummer wird das Ergebnis der Exklusiv-ODER-Operation invertiert, so daß gemeinsame Daten von benachbarten Tabellen in unterschiedlichen Bänken gespeichert sind.

[0141] Bei dem Beispiel, das in **Fig.** 14 gezeigt ist, entsprechen die Blöcke, die mit Block1 bis Block4 bezeichnet sind, jeweils dem oberen linken Quadranten, dem oberen rechten Quadranten, dem unteren linken Quadranten und dem unteren rechten Quadranten der Tabelle mit 512 × 512 Texeln. Für die Blöcke Block1 bis Block4 sind die Bits S[8], T[8] jeweils gleich [0,0], [1,0], [0,1] und [1,1]. Daher ist das Ergebnis der XOR-Operation S[8] XOR T[8] für den Block1 gleich null. Da die Tabelle eine ungeradzahlige Tabellennummer (d.h. neun) aufweist, wird die Inverse dieses Ergebnisses (die gleich eins ist) verwendet, um anzuzeigen, daß der Block1 in der Bank eins des Cache gespeichert werden soll. Für Block1 ist die Inverse des Ergebnisses der XOR-Operation S[8] XOR T[8] gleich null, was anzeigt, daß Block1 in der Bank null in dem Cache gespeichert werden soll. Für Block3 und Block4 ist die Inverse des Ergebnisses der XOR-Operation S[8] XOR T[8] eins bzw. null, was anzeigt, daß Block3 in der Bank eins gespeichert werden soll und Block4 in der Bank null gespeichert werden soll.

[0142] Für eine Tabelle mit einer Tabellennummer von zehn für die gleiche Textur, wie in dem Beispiel von **Fig.** 14 gezeigt ist, würde die Tabelle in sechzehn Blöcke von jeweils 256 × 256 Texeln geteilt werden, da die Tabelle 1024 × 1024 Texel groß ist. Für jeden Block würden die Ergebnisse von S[9] XOR T[9] die Banknummer für diesen speziellen Block anzeigen. Es sei bemerkt, daß das Ergebnis der XOR-Operationen für jeden Block der Tabelle, die eine Tabellennummer von zehn aufweist, nicht invertiert wird, wie

dies für die benachbarte Tabelle mit einer Tabellennummer neun der Fall war, so daß die entsprechenden Quadranten in den zwei Tabellen in unterschiedlichen Cache-Bänken gespeichert werden.

[0143] Abhängig von der Größe der Tabelle kann die Blockkennung für Texturdatenblöcke, die die Tabelle darstellen, zumindest ein S-Koordinatenbit hoher Ordnung und ein T-Koordinatenbit hoher Ordnung aufweisen, die den Ort des Blocks in der speziellen MIP-Tabelle anzeigen. Für eine MIP-Tabelle mit 512 × 512 Texeln mit einer Tabellennummer von neun wäre nur ein S-Koordinatenbit und ein T-Koordinatenbit in der Blockkennung erforderlich, um den Ort jedes Blocks in der MIP-Tabelle anzuzeigen. Für eine MIP-Tabelle mit 1024 × 1024 Texeln mit einer Tabellennummer von zehn, die sechzehn Datenblöcke aufweist, wären zwei S-Koordinatenbits und zwei T-Koordinatenbits in der Blockkennung erforderlich, um den Ort jedes Blocks in der MIP-Tabelle anzuzeigen. Für Tabellen mit einer Tabellennummer von acht oder kleiner sind keine S- und T-Bits in der Blockkennung erforderlich. Wenn MIP-Tabellen-Texturdaten aus dem Hauptspeicher des Hostcomputers in den Cache-Speicher heruntergeladen werden, decodiert der Rostcomputer die S- und T-Koordinatenbits höherer Ordnung der Blockkennung unter Verwendung des oben erläuterten Exklusiv-ODER-Schemas, um die spezielle Bank zu bestimmen, in die jeder Datenblock geschrieben werden soll.

[0144] Um Texturdaten derart zuzuordnen, daß ungebrauchter Speicherplatz minimiert ist, kann jeder Datenblock ferner in sechzehn Unterblöcke von 64 × 64 Texeln unterteilt sein. Jeder Texturdaten-Unterblock weist eine Unter-Textur-ID auf, die den Ort des speziellen Unterblocks in dem Block identifiziert.

[0145] Die Unter-Textur-ID weist zwei S-Bits S[1:0] und zwei T-Bits T[1:0] auf. Mehrere Unter-Texturen von einer oder mehreren MIP-Tabellen von einer oder mehreren Texturen können in einem einzelnen Block gespeichert werden.

[0146] **Fig.** 15 zeigt Block1 und Block1, die jeweils Bänken null und eins des Cache zugewiesen sind, welche in sechzehn Unter-Texturen der Größe 64 × 64 Texel unterteilt sind. Die Unter-Texturen jedes Blocks sind mit ST0 bis ST15 bezeichnet und durch eine Unter-Textur-ID, die zwei S-Koordinatenbits und zwei T-Koordinatenbits aufweist, identifiziert. Die Unter-Texturen weisen eine konsistente Bezeichnung, aber Spiegelorte in den zwei Cache-Bänken auf, um mit dem oben beschriebenen Speicherzuweisungsschema konsistent zu sein. Die Größe der Unter-Texturen von 64 × 64 Texeln ist exemplarisch ausgewählt und kann geändert werden. Beispielsweise würde eine kleinere Unter-Textur ermöglichen, daß mehr Texturen in die gleichen Blöcke gepackt werden. Es sollte offensichtlich sein, daß die Unter-Textur-ID mehr Bits einschließen müßte, wenn die Größe der Unter-Textur verringert wird.

[0147] Während der Aufbereitung werden für jeden Texelstrom, der interpoliert werden soll, die Textur-ID, die Unter-Textur-ID und ein 8-Bit-Wort, das die Größe der Basistabelle für diese Textur darstellt, welche diesen Texeln zugeordnet ist, durch die 3-D-Pipeline zu dem Kachelbilder/Grenzprüfer geliefert, der die Daten temporär in einem 20-Bit-Register (nicht gezeigt) speichert. Wenn ein Texel, das interpoliert werden soll, eine unterschiedliche Unter-Textur-ID oder Textur-ID aufweist, werden die neuen Daten zu dem Kachelbilder/Grenzprüfer geliefert und in dem Register gespeichert. Die Unter-Textur-ID kann als ein Teil der Texeladresse verwendet werden, wie nachfolgend erläutert wird.

[0148] Ob die Texeladresse S-, T-Koordinatenbits einer Unter-Textur-ID aufweist, hängt von der Größe der Tabelle ab, die adressiert wird, und der Größe der Basistabelle dieser Textur. Wenn die Tabelle, die adressiert wird, eine Tabellengröße von sieben oder kleiner aufweist, und die entsprechende Basistabelle derselben ebenfalls eine Größe von sieben oder kleiner aufweist, weisen bestimmte obere Adreßbits der Texeladresse Bits von der Unter-Textur-ID auf, um den Ort der Unter-Textur in dem Block zu adressieren, wie nachfolgend detailliert erläutert wird. Wie oben erklärt wurde sind, wenn die Basistabelle eine Tabellennummer von acht oder größer aufweist, die Orte aller MIP-Tabellenquadranten für diese Textur in ihren jeweiligen Datenblöcken vordefiniert. Wenn aus einer der Tabellen für diese Textur, die eine Tabellennummer von sieben oder kleiner aufweist, auf ein Pixel zugegriffen wird, sind daher diese vorbestimmten Orte bekannt und werden verwendet, um die oberen Bits der Texeladresse für jeden Quadranten zu erzeugen, ohne die Unter-Textur-ID zu verwenden. Wenn die Basistabelle einer Textur eine Tabellennummer von sieben oder kleiner aufweist, sind die Orte der MIP-Tabellenquadranten jedoch nicht vordefiniert, und die Unter-Textur-ID-Bits werden als die oberen Bits der Texeladresse verwendet, um den Ort der Unter-Textur zu bestimmen.

[0149] Wie oben erwähnt wurde, können mehrere Tabellen von unterschiedlichen Texturen in unterschiedlichen Unter-Texturen eines einzelnen Datenblocks gespeichert werden, solange die Basistabelle von dieser Textur klein genug ist. Wenn dies auftritt, weist die Texturadresse für jede Tabelle Unter-Textur-ID-Bits auf. Wenn beispielsweise vier unterschiedliche Tabellen, die Tabellennummern von sieben aufweisen, von vier unterschiedlichen Texturen unter unterschiedlichen Unter-Texturen in einem Block zugewiesen werden, und die Tabellennummer für die Basistabelle jeder Textur sieben ist, dann wäre ein S-Koordinatenbit und ein T-Koordinatenbit von der Unter-Textur-ID ein Teil der Texeladresse, um

zwischen den Texturen zu unterscheiden. Die Routine, durch die der Kachelbilder/Grenzprüfer die Texeladresse berechnet, wird nachfolgend bezugnehmend auf **Fig.** 17 beschrieben.

[0150] Bei dem dargestellten Ausführungsbeispiel der Erfindung werden MIP-Tabellen-Texturdaten Block für Block heruntergeladen. Es sollte jedoch offensichtlich sein, daß alternativ eine Unter-Textur-ID in der Blockkennung enthalten sein kann, so daß Unter-Texturen aus dem Hauptspeicher heruntergeladen werden könnten. Ferner sind die Größen der Blöcke und Unter-Texturen, die bei diesem Ausführungsbeispiel beschrieben sind, nur dazu bestimmt, exemplarisch zu sein und können geändert werden, um sich für eine beliebige Anwendung zu eignen.

VI. Cache-Blockkennung und -Blockindex

[0151] Das Cache-Verzeichnis weist eine Blockkennung für jeden seiner vierundsechzig Einträge auf und identifiziert einen entsprechenden Blockindex für jeden Eintrag. Der Blockindex identifiziert den physikalischen Ort in dem Cache, an dem der Anfang des entsprechenden Texturdatenblocks gespeichert ist. Die Blockkennung ist ein 23-Bit-Identifizierer, der jeden Texturdatenblock auf die Art und Weise, die in **Fig.** 16 gezeigt ist, eindeutig identifiziert.

[0152] Um jedes Texel von Texturdaten eindeutig zu identifizieren, muß die Textur, der dasselbe zugeordnet ist, identifiziert werden. Bei einem Ausführungsbeispiel der Erfindung unterstützt die Texturabbildungshardware eine 8-Bit-Textur-ID, die eine Textur eindeutig identifiziert. Außerdem wird für Texturdaten von unterschiedlichen Texturen, die in dem gleichen Block gespeichert sind, eine zusätzliche 4-Bit-Unter-Textur-ID durch die Hardware unterstützt, um die Texturen zu identifizieren. Folglich unterstützt die Texturabbildungshardware der vorliegenden Erfindung 2^{12} oder 4096 eindeutige Texturen, die zu jeder Zeit aktiv sein können.

[0153] Wie oben erläutert wurde, ist jede Textur durch eine Reihe von MIP-Tabellen dargestellt, wobei bei einem Ausführungsbeispiel der Erfindung jede der MIP-Tabellen mit einer Tabellennummer versehen ist, die die Position derselben in der Reihe von MIP-Tabellen anzeigt. Folglich ist jedes Datentexel nicht nur durch die Textur-ID, die Unter-Textur-ID und die Größe der Basistabelle für diese Textur identifiziert, sondern ferner durch die Tabellennummer der MIP-Tabelle, der dieselbe zugeordnet ist. Schließlich ist das Texel durch die S- und T-Koordinaten (d.h. den interpolierten S-, T-Wert desselben) eindeutig in der MIP-Tabelle identifiziert.

[0154] Anders als die Unter-Textur-ID und die Texturtabellen-Basisgröße werden die oben beschriebenen Parameter, die ein Texel eindeutig identifizieren, verwendet, um die 23-Bit-Blockkennung zu erzeugen. Bezüglich der Tabellennummer und der Koordinaten S und T ist bei einem Ausführungsbeispiel der vorliegenden Erfindung die Hardware, die verwendet ist, um die Koordinaten S und T zu erzeugen, auf fünfzehn Bits begrenzt. Daher weist bei diesem Ausführungsbeispiel die größte Texturtabelle, die durch die Hardware unterstützt wird, ein 15-Bit-S-Feld [14:0] und ein 15-Bit-T-Feld [14:0] auf, was eine maximale Texturtabelle zur Folge hat, die 32000 × 32000 (32 K × 32 K) Texel groß ist. Wie oben erläutert wurde, weist jeder Texeldatenblock 256 × 256 Texel auf. Folglich werden die S- und T-Bits niedriger Ordnung (d.h. T[7:0] und S[7:0]) verwendet, um ein spezielles Texel in einem Texeldatenblock zu identifizieren. Nur die S- und T-Bits (T[14:8] und S[14:8]) werden in der Blockkennung verwendet, um einen speziellen Texeldatenblock zu identifizieren.

[0155] Wie oben erwähnt wurde ist jeder MIP-Tabelle eine Tabellennummer zugewiesen, die dieselbe eindeutig in der Reihe von Tabellen für ihre entsprechende Textur identifiziert. Ungeachtet der Anzahl von MIP-Tabellen in der Reihe von Tabellen für eine Textur ist die kleinste MIP-Tabelle in der Reihe (d.h. ein Texel groß) zugewiesen, um die Tabelle mit der Nummer null zu sein. Da die größte Reihe von MIP-Tabellen für eine 32K × 32K-Textur sechzehn MIP-Tabellen aufweist, ist die größte unterstützte Tabellennummer fünfzehn.

[0156] Die Art und Weise, auf die die Blockkennung gebildet wird, ist in der Tabelle von **Fig.** 16 gezeigt. Die acht Bits hoher Ordnung [22:15] der Blockkennung entsprechen der Textur-ID der Textur, die durch den Texturdatenblock dargestellt ist. Die Bits der Blockkennung niederer Ordnung [13:00] entsprechen den T- und S-Koordinaten hoher Ordnung, T[14:08] und S[14:08]. Die Blockkennung [14] entspricht einem Tabellenbit, das in Verbindung mit den Werten in dem T-Koordinatenfeld hoher Ordnung die Identifizierung der Tabellennummer ermöglicht. Es sollte offensichtlich sein, daß Tabellen, die kleiner als das Maximum von 32K × 32K sind, keine vollen S- und T-Adreßfelder verwenden, derart, daß, je kleiner die Tabelle ist, desto mehr S- und T-Adreßbits hoher Ordnung unbenutzt bleiben. Wie in **Fig.** 16 gezeigt ist, ist für Tabellen, die eine Tabellennummer größer als acht aufweisen, das Blockkennungsbit, das dem niederstwertigen unverwendeten T-Koordinatenbit entspricht, auf eine logische "0" gesetzt, und die Blockkennungsbits, die den verbleibenden T-Koordinatenbits hoher Ordnung entsprechen, und das Tabellenbit sind auf eine logische "1" gesetzt. Für eine Tabellennummer fünfzehn, die alle T-Koordinatenbits verwendet, ist das Tabellenbit auf eine logische "0" gesetzt. Durch das Lesen der Blockkennungsbits [14:07], die dem Tabellenbit und den T-Koordinatenbits [14:8] hoher Ordnung entsprechen, zeigt die Position der ersten logischen "0", die beim Lesen von links nach

rechts angetroffen wird, die Tabellennummer an, die durch die Blockkennung dargestellt ist. Wenn alle Blockkennungsbits [14:08] eine logische "1" aufweisen, dann sind Tabellennummern von acht und kleiner dargestellt.

[0157] Wie oben beschrieben wurde, werden alle Tabellen einer speziellen Textur mit einer Tabellennummer von acht oder kleiner in zwei Datenblöcken gespeichert, wobei sich jeder Block in einer unterschiedlichen Bank des Cache befindet. Zwei Quadranten, oder eine Hälfte, von jeder der Tabellen, die Tabellennummern von acht und kleiner aufweisen, sind in jedem der zwei Blöcke gespeichert. Das Blockkennungsbit [07] stellt dar, in welchem der zwei Blöcke jeder Halb-Abschnitt der Tabellen, die Tabellennummern von acht und kleiner aufweisen, gespeichert ist. Folglich weist für jede der Tabellen, die eine Tabellennummer von acht oder kleiner aufweisen, das Blockkennungsbit [07] einen Wert von "0" für die eine Hälfte (zwei Quadranten) der Tabelle (die in dem Bank0-Block gespeichert ist) auf und weist einen Wert von "1" für die andere Hälfte (zwei Quadranten) dieser Tabelle (die in dem Bank1-Block gespeichert ist) auf. Es sollte offensichtlich sein, daß, da alle Tabellen von einer speziellen Textur, die eine Tabellennummer von acht oder kleiner aufweisen, in zwei Blöcken gespeichert sind, nur ein Blockkennungsbit verwendet ist, um diese zwei Blöcke zu identifizieren. Die spezielle Tabellennummer für jede der Tabellen mit einer Nummer von acht oder kleiner ist daher nicht als ein Teil des Blockkennungsfeldes gespeichert.

[0158] Der Wert des Blockkennungsbits [07] für jeden Quadranten jeder der Tabellen, die eine Tabellennummer von acht oder kleiner aufweisen, wird basierend auf dem Schema zum Bestimmen der Bank, in der der Quadrant gespeichert werden soll, berechnet. Dieses Schema weist die logische Exklusiv-ODER-Operation der Werte der MSB-Bits für jeden Quadranten von geradzahlig numerierten Tabellen und der Inversen der Operation für jeden Quadranten von ungeradzahlig numerierten Tabellen auf.

[0159] Wie in **Fig.** 16 gezeigt ist, sind die Blockkennungsbits [6:0], die den S-Adreßbits hoher Ordnung entsprechen, für kleine Tabellen auf eine logische "0" gesetzt, wenn die S-Adreßbits ungenutzt sind, so daß, wenn irgendeines dieser Bits als eine logische "1" in Verbindung mit einer Tabellennummer, die anzeigt, daß dieses gleich einer logischen "0" sein sollte, erfaßt wird, dasselbe verwendet werden kann, um anzuzeigen, daß keine gültigen Daten in dem Cache-Verzeichniseintrag enthalten sind.

[0160] Wie oben erläutert wurde, diktieren für jeden MIP-Tabellenquadranten die Ergebnisse einer logischen Exklusiv-ODER-Operation (XOR-Operation) auf den Werten der höchstwertigen S- und T-Koordinaten für den Quadranten die SDRAM-Bank in dem Cache, der der Quadrant zugewiesen ist. Die Banknummer ist für Tabellen, die eine geradzahlige Tabellennummer aufweisen, gleich dieser XOR-Operation, und ist für Tabellen, die eine ungeradzahlige Tabellennummer aufweisen, gleich der logischen Inversen der XOR-Operation. Dies ist in der rechten Spalte der Tabelle von **Fig.** 23 gezeigt, in der das Symbol "^" eine XOR-Operation und das Symbol "!" die logische Inverse anzeigt. Für Tabellen mit einer Tabellennummer von neun oder größer verbraucht jeder Quadrant zumindest einen vollen Datenblock, wobei jeder Block in der Bank gespeichert wird, die durch die XOR-Operation, die in der letzten Spalte der Tabelle von **Fig.** 16 gezeigt ist, diktiert wird.

[0161] Für Tabellen mit einer Tabellennummer von acht oder kleiner besetzen alle diese Tabellen zwei Datenblöcke, einen Block in jeder Bank. Die letzten zwei Zeilen der Tabelle von **Fig.** 16 entsprechen unterschiedlichen Hälften (zwei Quadranten) jeder Tabelle, die eine Tabellennummer von acht oder kleiner aufweist. Das Blockkennungsbit [07] zeigt, in welchem von dem Bank0-Block oder Bank1-Block die halbe Tabelle gespeichert ist. Der Wert dieses Bits [07] wird basierend auf der beschriebenen XOR-Operation berechnet. Beispielsweise wäre für eine Tabelle mit einer Tabellennummer acht das Blockkennungsbit [07] für jeden Quadranten der Tabelle gleich S[7] XOR T[7]. Für jeden Quadranten einer Tabelle mit einer Tabellennummer von sieben wäre das Blockkennungsbit [07] gleich der Inversen von S[6] XOR T[6]. Das Blockkennungsbit [07] wird in gleicher Weise für jeden Quadranten der kleineren Tabellen berechnet, wobei das Ergebnis der XOR-Operation nur für ungeradzahlig numerierte Tabellen invertiert wird. Aus dem vorhergehenden sollte offensichtlich sein, daß, da zwei Quadranten jeder Tabelle (die eine Tabellennummer von acht oder kleiner aufweist) in dem gleichen Block gespeichert sind, diese zwei Quadranten jeder Tabelle das gleiche Blockkennungsbit [07] aufweisen würden.

[0162] Wenn ein Treffer zwischen interpolierten S-, T-Koordinaten (die ein Texel adressieren, auf das zugegriffen werden soll) auftritt, und eine der 23-Bit-Blockkennungen in dem Cache-Verzeichnis ist, erzeugt das Cache-Verzeichnis einen Blockindex, der den physikalischen Ort in dem Cache-Speicher identifiziert, an dem der Cache-Block, der das Texel enthält, gespeichert ist. Der Cache speichert vierundsechzig Texeldatenblöcke auf einmal. Daher ist, um eine Blockadresse in dem Cache-Speicher zu identifizieren, ein 6-Bit-Blockindex (26 = 64) vorgesehen, der als die Adreßbits hoher Ordnung zu dem Cache dient, wie oben beschrieben ist.

[0163] Die Texeladresse ist ein 16-Bit-Wort, das die Bits S[7:0] und T[7:0] aufweist, die den Ort des Texels anzeigen, auf das in dem Block von 256 × 256 Texeln

zugegriffen werden soll. Die Texeladresse wird aus den interpolierten S-, T-Koordinaten, der Tabellennummer der Tabelle, auf die zugegriffen werden soll, den Textur- und Unter-Textur-IDs und der Basistabellengröße der Textur gemäß einer Routine, die nachfolgend bezugnehmend auf **Fig.** 17 erläutert wird, berechnet. Wie oben erläutert wurde, werden das LSB-S-Bit und das LSB-T-Bit der Texeladresse decodiert, um die entsprechende Verschachtelung, in der das Texel gespeichert ist, zu bestimmen. Die verbleibenden vierzehn Bits der Texeladresse dienen in Verbindung mit den sechs Bits des Blockindex als die Cache-Adresse (wobei die sechs Bits des Blockindex die sechs MSBs der Cache-Adresse sind), die dem SDRAM-Paar in der decodierten Verschachtelung des Cache geliefert werden.

VII. Texeladreßberechnung

[0164] Während der Aufbereitung empfängt das Kachelbilder/Grenzprüfer-Element **72** den interpolierten S-, T-Wert des Texels, auf das zugegriffen werden soll, ebenso wie ein 4-Bit-Wort, das die Tabellennummer der Tabelle darstellt, aus der auf das Texel zugegriffen werden soll, von dem Parameterinterpolator **64**. Jeder der interpolierten S- und T-Koordinatenwerte, die von dem Parameterinterpolator empfangen werden, weist sechzehn Ganzzahl-Bits und acht Bruchteil-Bits auf. Das 4-Bit-Wort, das die Tabellennummer darstellt, schließt Tabellen in dem Bereich von der Tabellennummer null (ein Texel groß) zu der Tabellennummer fünfzehn (32K × 32K Texel groß) ein und wird, wie oben beschrieben wurde, aus dem Gradienten berechnet. Ein Vergleich des interpolierten S-, T-Werts mit den Blockkennungseinträgen in dem Cache-Speicher wird dann durchgeführt. Wenn ein Treffer mit einer der Blockkennungen stattfindet, wird der Blockindex erzeugt. Zur gleichen Zeit, zu der die Cache-Verzeichnissuche durchgeführt wird, wird die Texeladresse gemäß der Routine, die nachfolgend bezugnehmend auf das Flußdiagramm von **Fig.** 17 beschrieben wird, berechnet.

[0165] Die Texeladresse wird basierend auf der Textur-ID, der Unter-Textur-ID, der Tabellennummer, der Basistabellennummer und der interpolierten S-, T-Koordinaten des Texels durch den Kachelbilder/Grenzprüfer berechnet. Der Kachelbilder/Grenzprüfer besitzt alle diese Informationen. Für jedes einzelne Texel, auf das zugegriffen werden soll, empfängt der Kachelbilder/Grenzprüfer die interpolierten S-, T-Koordinaten (einschließlich sechzehn Ganzzahl- und acht Bruchteil-Bits sowohl für S als auch für T), ebenso wie ein 4-Bit-Wort, das die Tabellennummer darstellt, aus der auf das Texel zugegriffen werden soll, von dem Parameterinterpolator. Zusätzlich wird durch die 3-D-Pipeline (die ebenfalls durch den Parameterinterpolator läuft) ein Befehl empfangen, der die 8-Bit-Textur-ID, die 4-Bit-Unter-Textur-ID und ein 8-Bit-Wort aufweist, das die Größe der Basistabelle für diese Textur darstellt. Das 8-Bit-Wort, das die Größe der Basistabelle darstellt, weist vier S-Bits und vier T-Bits auf, die dem Tabellennumerierungsschema der Erfindung entsprechen und jeweils die Größe der S-Dimension und der T-Dimension der Basistabelle definieren. Beispielsweise kann jedes der 4-Bit-S- und -T-Worte einen Wert in einem Bereich von null (was einer Dimension von einem Texel entspricht) bis fünfzehn (was einer Dimension von 32K Texeln entspricht) aufweisen. Die 20 Datenbits, die die Textur-ID, die Unter-Textur-ID und die Basistabellennummer enthalten, werden temporär in einem 20-Bit-Register (nicht gezeigt) in dem Kachelbilder/Grenzprüfer gespeichert, bis dieselben mit neuen und unterschiedlichen Daten für ein nachfolgendes Texel, auf das aus dem Cache zugegriffen werden soll, ersetzt werden. Mit diesen Informationen berechnet der Kachelbilder/Grenzprüfer die Texeladresse für jedes Texel.

[0166] Wie oben erklärt wurde, weisen für Texturen, die eine Basistabelle mit einer Tabellennummer größer oder gleich acht aufweisen (was einer Basistabelle von 256 × 256 Texeln oder größer entspricht), die Quadranten jeder Tabelle in dieser Textur einen vordefinierten Ort in den Texturdatenblöcken und Cache-Speicherbänken auf. Folglich kann jedes Bit der Texeladresse für jedes Texel einer solchen Textur gemäß diesem bekannten vordefinierten Zuweisungsschema berechnet werden. Für Texturen, die eine Basistabelle mit einer Tabellennummer von sieben oder kleiner aufweisen (was einer Basistabelle von 128 × 128 Texeln oder kleiner entspricht), ist jedoch eine Anzahl von unterschiedlichen Speicherorten für jeden Quadranten der Tabelle dieser Textur verfügbar, weshalb bestimmte höherwertige Bits der Texeladresse einige oder alle Bits (oder die Inversen dieser Bits) der Unter-Textur-ID einschließen werden.

[0167] Die Routine, die durch den Kachelbilder/Grenzprüfer implementiert ist, um die Texeladresse zu berechnen, ist durch das Flußdiagramm von **Fig.** 17 gezeigt. Die Routine erfordert zum Vollenden nur einen Zyklus. Die Routine kann durch einen Satz von logischen Gattern (nicht gezeigt) implementiert sein, die den Grenzprüferabschnitt des Texturabbildungschips bilden. Es sollte für Fachleute offensichtlich sein, wie die logischen Gatter zu implementieren sind, um die Routine durchzuführen, die durch das Flußdiagramm von **Fig.** 17 umrissen ist. Beispielsweise kann die Routine in einer Softwaresimulationssprache, beispielsweise Verflog, geschrieben sein und dann durch ein Synthesewerkzeug, beispielsweise Synopsys™, das auf einem Universalprozessor arbeitet, in eine Logikgatterschaltung umgewandelt werden. Die Routine kann alternativ softwaremäßig geschrieben werden und durch einen Prozessor ausgeführt werden.

[0168] Die Routine beginnt bei einem Schritt **250**, in

dem die Texeladreßbits S[7:0], T[7:0] voreingestellt werden, um gleich den interpolierten S-, T-Koordinatenbits S[7:0], T[7:0] zu sein. Jedes der Bits der Texeladresse wird auf dem Wert bleiben, auf den dasselbe in diesem Schritt voreingestellt ist (gleich der entsprechenden S- oder T-Koordinate), ausgenommen eines Rücksetzens später in der Routine. Dann springt die Routine zu einem Schritt **252**, in dem bestimmt wird, ob die spezielle Tabelle, in der das Texel, das interpoliert wird, gespeichert ist, eine Tabellennummer größer oder gleich acht aufweist. Wenn dies der Fall ist, endet die Routine für ein solches Texel und die Bitwerte für die Texeladresse bleiben gemäß der Voreinstellung gleich den interpolierten S-, T-Koordinaten.

[0169] Wenn die Tabellennummer nicht größer oder gleich acht ist, springt die Routine zu einem Schritt **254**, in dem bestimmt wird, ob das Texel in der Bank mit der Nummer eins oder der Bank mit der Nummer null gespeichert ist. Wie oben beschrieben wurde, ist es durch das Untersuchen des Werts des Blockkennungsbits [07] bekannt, ob das Texel in der Bank mit der Nummer eins oder in der Bank mit der Nummer null gespeichert ist.

[0170] Wenn das Texel in der Bank mit der Nummer null gespeichert ist, springt die Routine zu einem Schritt **256**, in dem bestimmte Texeladreßbits aus ihren voreingestellten Werten rückgesetzt werden. Für Tabellen mit Tabellennummern von eins bis vier: Texeladreßbit S[4] = 1, und für Tabellen mit Tabellennummern von eins und zwei: Texeladreßbit S[2] = 1. Wenn das Texel in der Bank null gespeichert ist, springt die Routine zu einem Schritt **258**, in dem für Tabellen mit Tabellennummern von null bis fünf: Texeladreßbit S[5] = 1, für Tabellen mit Tabellennummern von null bis drei: Texeladreßbit S[3] = 1, und für Tabellen mit Tabellennummern von null und eins: Texeladreßbit S[1] = 1.

[0171] Von jedem der Schritte **256** und **258** springt die Routine zu einem Schritt **260**, in dem bestimmt wird, ob die Basistabelle eine Tabellennummer aufweist, die größer als oder gleich acht ist. Wenn dies der Fall ist, springt die Routine zu einem Schritt **262**, in dem bestimmt wird, ob das Texel in der Bank null oder der Bank eins gespeichert ist. Wenn das Texel in der Bank eins gespeichert ist, springt die Routine zu einem Schritt **264**, in dem für eine Tabelle mit einer Tabellennummer von sieben: Texeladreßbit S[7] = 0, und für Tabellen mit Tabellennummern null bis sechs: Texeladreßbits S[7:6] = 0:1. Die Routine ist dann für ein solches Texel beendet. Für ein Texel, das in der Bank null gespeichert ist, springt die Routine zu einem Schritt **266**, in dem für eine Tabelle mit einer Tabellennummer von sieben: Texeladreßbit S[7] = 1, und für Tabellen mit Tabellennummern von null bis sechs: Texeladreßbits S[7:6] = 1:0. Die Routine ist dann für ein solches Texel beendet.

[0172] Wenn die Basistabelle keine Tabellennummer größer als oder gleich acht aufweist, springt die Routine zu einem Schritt **268**, in dem bestimmt wird, ob die Basistabelle eine Tabellennummer, die gleich sieben ist, aufweist. Wenn dies der Fall ist, springt die Routine zu einem Schritt **270**, in dem bestimmt wird, ob das Texel in der Bank null oder eins gespeichert ist. Wenn das Texel in der Bank eins gespeichert ist, springt die Routine zu einem Schritt **272**, in dem für eine Tabellennummer sieben das Texeladreßbit S[7] gleich der Inversen des Unter-Textur-ID-Bits S[1] gesetzt wird und das Texeladreßbit T[7] gleich dem Unter-Textur-ID-Bit T[1] gesetzt wird, und in dem für Tabellen mit Tabellennummern von null bis sechs die Texeladreßbits S[7:6] gleich der Inversen des Unter-Textur-ID-Bits S[1] bzw. eins gesetzt werden und das Texeladreßbit T[7] gleich dem Unter-Textur-ID-Bit T[1] gesetzt wird. Die Routine endet dann für ein solches Texel. Wenn das Texel in der Bank null gespeichert ist, springt die Routine zu einem Schritt **274**, in dem für eine Tabelle mit einer Tabellennummer sieben das Texeladreßbit S[7] gleich dem Unter-Textur-ID-Bit S[1] gesetzt wird und das Texeladreßbit T[7] gleich dem Unter-Textur-ID-Bit T[1] gesetzt wird, und in dem für Tabellen mit Tabellennummern von null bis sechs die Texeladreßbits S[7:6] gleich dem Unter-Textur-ID-Bit S[1] bzw. null gesetzt werden, und das Texeladreßbit T[7] gleich dem Unter-Textur-ID-Bit T[1] gesetzt wird. Dann endet die Routine für ein solches Texel.

[0173] Wenn die Basistabelle der Textur weder eine Tabellennummer, die größer oder gleich acht ist (was im Schritt **260** bestimmt wird), noch eine Tabellennummer, die gleich sieben ist (was im Schritt **268** bestimmt wird), aufweist, dann ist selbstverständlich bekannt, daß die Basistabelle der Textur eine Tabellennummer aufweist, die kleiner als oder gleich sechs ist, und die Routine springt zu einem Schritt **276**, in dem bestimmt wird, ob das Texel in der Bank null oder der Bank eins gespeichert ist. Wenn das Texel in der Bank eins gespeichert ist, springt die Routine zu einem Schritt **278**, in dem die Texeladreßbit S[7:6] gleich der Inversen den Unter-Textur-ID-Bits S[1:0] gesetzt werden und die Texeladreßbits T[7:6) gleich dem Unter-Textur-ID-Bits T[1:0) gesetzt werden. Die Routine ist dann für ein solches Texel abgeschlossen. Wenn das Texel in der Bank null gespeichert ist, springt die Routine zu einem Schritt **280**, in dem die Texeladreßbits S[7:6] gleich den Unter-Textur-ID-Bits S[1:0] gesetzt werden und die Texeladreßbits T[7:6] gleich den Unter-Textur-ID-Bits T[1:0] gesetzt werden. Die Routine ist dann für ein solches Texel abgeschlossen.

VIII. Texturdatenorganisations-Beispiele

[0174] sDas folgende Beispiel beschreibt die Prozedur, durch die der Hostcomputer Texturdaten gemäß dem oben beschriebenen Ausführungsbeispiel der

Erfindung organisiert. Für eine spezielle Anwendung kann ein Grundelement A, das aufbereitet werden soll, auf eine Textur A abbilden, und ein Grundelement B kann auf eine Textur B abbilden. Eine Möglichkeit für den Hostcomputer bestünde darin, die Textur A in eine Mehrzahl von Texturdatenblöcken zu organisieren, und danach die Textur B in verschiedene Unter-Texturen in den gleichen Blöcken wie die Textur A zu organisieren. Der Hostcomputer würde die Texturdatenblöcke, die die Texturen A und B einschließen, in den Cache-Speicher herunterladen, bevor die Grundelemente A und B aufbereitet werden.

[0175] Alternativ kann der Host die Textur A in eine Mehrzahl von Texturdatenblöcken organisieren und dann die Blöcke, die die Textur A aufweisen, in den Cache-Speicher herunterladen. Der Hostcomputer könnte dann die Textur B in verschiedenen Unter-Texturen in den gleichen Blöcken wie die Textur A in dem Hauptspeicher organisieren. In dieser Situation würde der Hostcomputer einen Befehl erteilen, um die Operation des Texturabbildungschips **46** (**Fig.** 2) anzuhalten, und würde die neu organisierten Texturdatenblöcke (die die Texturen A und B in den gleichen Blöcken aufweisen) in den Cache-Speicher des Texturabbildungssystems herunterladen. Es sollte offensichtlich sein, daß, wenn die HALT-Bedingung nicht implementiert wäre und die neu organisierten Daten nicht aus dem Hauptspeicher in den Cache-Speicher des Texturabbildungssystems heruntergeladen würden, während der Aufbereitung des Grundelements B auf falsche Texturabbildungsdaten zugegriffen werden könnte. Dies ist der Fall, da, wenn das Grundelement B aufbereitet wird, ein Treffer in dem Cache-Verzeichnis auftreten würde, da die Lese-Cache-Kennung für den Datenblock, der die Textur B aufweist, mit der Blockkennung, die den Datenblöcken in dem Cache entspricht, die die Textur A speichern, übereinstimmen würde. Jedoch speichern die Datenblöcke in dem Cache nur Texturdaten, die sich auf die Textur A beziehen, nicht auf die Textur B.

IX. Umgehung der dreidimensionalen Grundelementpipeline und Unterbrechungsschema für das Herunterladen von Texturtabellen

[0176] Wie oben erläutert wurde, ermöglicht ein Merkmal der vorliegenden Erfindung, daß eine MIP-Tabelle für eine neue Textur durch einen Datenweg, der von der Pipeline für das Handhaben der 3-D-Grundelementdaten getrennt ist, in den Lokalspeicher in der Texturabbildungshardware heruntergeladen wird. Bezugnehmend auf das veranschaulichende Ausführungsbeispiel, das in den Figuren offenbart ist, weisen die Texturabbildungsplatine **12** (**Fig.** 2) und der Texturabbildungschip **46** (**Fig.** 3) jeweils getrennte Tore auf, um jeweils 3-D-Grundelementdaten und Texturdaten zu empfangen. Die 3-D-Grundelementdaten werden über den Bus **18** von dem Konzentratorchip **36** empfangen, wohingegen die Texturdaten über den Bus **24** von dem 2-D-Geometrie-Beschleunigerchip **34** empfangen werden. Wenn neue Texturdaten von dem Hostcomputer **15** in den Texturabbildungschip **46** heruntergeladen werden, muß daher die 3-D-Grundelementpipeline durch die Eingangsplatine **10** und den Texturabbildungschip **46** nicht geräumt werden, wodurch verglichen mit herkömmlichen Texturabbildungssystemen, die ein Räumen der 3-D-Grundelementpipeline erfordern, wann immer neue Texturdaten in den Lokalspeicher in der Texturabbildungshardware heruntergeladen werden, eine erhöhte Bandbreite geliefert wird.

[0177] Der getrennte Datenweg für das Herunterladen von Texturdaten, der die 3-D-Grundelementpipeline umgeht, ist besonders vorteilhaft in Verbindung mit dem oben beschriebenen Ausführungsbeispiel der vorliegenden Erfindung, bei dem der Lokalspeicher auf der Texturabbildungsplatine **12** als ein Cache implementiert ist. Wie oben erläutert wurde, wird, wenn neue Texturdaten in den Cache heruntergeladen werden, nur der erforderliche Teil der MIP-Tabelle heruntergeladen, und nicht die gesamte Reihe von MIP-Tabellen für die Textur. Folglich ermöglicht es die 3-D-Pipeline-Umgehung, daß Fehlzugriffe ohne ein Räumen der Pipeline gehandhabt werden.

[0178] Wie oben erläutert wurde, sind bei einem Ausführungsbeispiel der Erfindung, das in **Fig.** 2A gezeigt ist, Teile des Grafiksystems dupliziert, um die Systembandbreite zu erhöhen. Die Texturabbildungsplatine **12** ist mit zwei Texturabbildungschips **46A** und **46B** und zwei Cache-Speichern **48A** und **48B** versehen. Bei diesem Ausführungsbeispiel halten die beiden Cache-Speicher **48** zu allen Zeiten die gleichen Texturdaten, da beide Texturabbildungschips typischerweise unter Verwendung der gleichen Texturdaten auf Grundelementen arbeiten. Daher konserviert dieses Ausführungsbeispiel der vorliegenden Erfindung durch das Aktualisieren beider Caches, immer wenn ein Fehlzugriff von einem empfangen wird, die Systembandbreite, indem sichergestellt wird, daß die gleichen Texturdaten bei getrennten Operationen nicht in die zwei Caches heruntergeladen werden müssen.

[0179] Bei dem Dualausführungsbeispiel des Texturabbildungschips von **Fig.** 2A wird jeder Cache-Speicher nur mit Texturdaten, die von dem Hostcomputer heruntergeladen werden, aktualisiert und wird nicht lokal von der Texturabbildungshardware beschrieben. Daher ist eine Konsistenz zwischen den zwei Cache-Speichern beibehalten, indem sichergestellt ist, daß, wann immer Texturdaten von dem Hostcomputer als Reaktion auf einen Fehlzugriff von einem der Caches heruntergeladen werden, beide Caches mit den neuen Texturdaten aktualisiert werden. Wenn ein Cache-Fehlzugriff von einem der Texturabbildungschips **46** auftritt und eine Unterbre-

chung erzeugt wird, werden beide Texturabbildungschips **46** angehalten, so daß beide Cache-Speicher mit den heruntergeladenen Texturdaten aktualisiert werden können. Folglich spricht jeder Texturabbildungschip auf die Erzeugung eines Cache-Fehlzugriffsignals von einem beliebigen der Texturabbildungschips an, um den Betrieb anzuhalten. Außerdem unterstützt die vorliegende Erfindung gleichzeitige Cache-Fehlzugriffe von den zwei Texturabbildungschips **46** auf unterschiedliche Cache-Blöcke und spricht als Reaktion auf die Fehlzugriffe an durch das Herunterladen beider neuen Texturdatenblöcke in beide Caches.

[0180] Bei dem illustrativen Ausführungsbeispiel, das in **Fig.** 2 gezeigt ist, wird das Umgehen der 3-D-Grundelementpipeline durch die Verwendung der 2-D-Grundelementpipeline durch den 2-D-Geometrie-Beschleunigerchip **34**, um Texturdaten herunterzuladen, erreicht. Es sollte offensichtlich sein, daß der Datenweg für das Herunterladen von Texturdaten in den Texturabbildungschip **46** auf eine Anzahl von alternativen Arten implementiert sein kann, während noch die 3-D-Grundelementpipeline umgangen wird. Beispielsweise kann ein dazu bestimmter Datenweg von dem Hostcomputer zu der Texturabbildungsplatine vorgesehen sein.

[0181] Der Hostcomputer des Grafiksystems der vorliegenden Erfindung kann ein Betriebssystem wie z.B. UNIX verwenden, das mehrere Prozesse gleichzeitig verarbeiten kann, und das ein bestimmtes Schema liefert, um zu ermöglichen, daß ein Prozeß bestimmte Systembetriebsmittel verriegelt, derart, daß ein Prozeß nicht unterbrochen werden kann, wenn er verriegelt ist. Unter Verwendung dieses Verriegelungsschemas kann ein Prozeß, der bestimmte Hardwarebetriebsmittel verwendet, sicherstellen, daß der Prozeß nicht ausgelagert wird (swapped out), bis er diese Betriebsmittel entriegelt.

[0182] Bei einem Ausführungsbeispiel der Erfindung sind zwei Verriegelungstypen zur Verwendung durch die Prozesse vorgesehen, d.h. eine schnelle Verriegelung und eine langsame Verriegelung. Wenn eine schnelle Verriegelung verwendet ist, überprüft ein Prozeß, der eingelagert (swapped in) wird, die entsprechenden Hardwarebetriebsmittel, um zu bestimmen, ob er der letzte Prozeß war, um diese Betriebsmittel zu verwenden. Wenn er es war, fährt der Prozeß einfach fort, ohne den Zustand dieser Betriebsmittel wiederherzustellen. Wenn der Prozeß jedoch nicht der letzte war, um die Betriebsmittel zu benutzen, wird eine langsame Verriegelung angefordert, die die Wiederherstellung der Hardwarebetriebsmittel auf den Zustand, in dem dieselben waren, als der Prozeß zuletzt ausgelagert wurde, zur Folge hat. Es sollte offensichtlich sein, daß eine Anzahl alternativer Techniken verwendet werden kann, um die gleichen Ergebnisse zu erhalten.

[0183] Bei dem Ausführungsbeispiel der vorliegenden Erfindung, bei dem die 2-D-Grundelementpipeline verwendet ist, um Texturdaten herunterzuladen, während 3-D-Grundelemente aufbereitet werden, werden 2-D- und 3-D-Prozesse nicht gleichzeitig ausgeführt. Diese Beschränkung ist erfüllt, indem durch die Verwendung des Verriegelungsschemas, das durch das Betriebssystem des Hostcomputers geliefert wird, sichergestellt wird, daß kein 2-D-Prozeß beginnt, es sei denn, die 3-D-Pipeline ist leer, und daß kein 3-D-Prozeß beginnt, es sei denn, die 2-D-Pipeline ist leer. Wenn ein 3-D-Prozeß beginnt, aktiviert derselbe eine Verriegelung, während, wenn der vorhergehende Prozeß ein 2-D-Prozeß war, derselbe wartet, bis die 2-D-Pipeline leer ist, bevor er beginnt. In gleicher Weise aktiviert, wenn ein 2-D-Prozeß beginnt, derselbe eine Verriegelung, wobei, wenn der vorhergehende Prozeß ein 3-D-Prozeß war, derselbe wartet, bis die 3-D-Pipeline leer ist, bevor er beginnt.

[0184] Einige Prozesse führen sowohl 3-D- als auch 2-D-Operationen durch, und können zwischen 3-D-Grundelementen und 2-D-Grundelementen schalten, ohne die langsame Verriegelung aufzugeben. Derartige Prozesse implementieren ferner ein Schema, um sicherzustellen, daß die 3-D-Pipeline leer ist, bevor 2-D-Grundelementdaten in die Hardware heruntergeladen werden, und um in gleicher Weise sicherzustellen, daß die 2-D-Pipeline leer ist, bevor 3-D-Grundelementdaten heruntergeladen werden. Um dieses Ergebnis zu erreichen, können Registerstatusbits vorgesehen sein, die anzeigen, ob jede der 2-D- und 3-D-Grundelementpipelines leer ist. Jeder Prozeß, der sowohl 2-D- als auch 3-D-Grundelementdaten verwendet, liest dieses Statusregisterbit, um sicherzustellen, daß die Pipelines leer sind, bevor er zwischen den 2-D- und 3-D-Grundelementdaten schaltet.

[0185] Es sollte offensichtlich sein, daß, obwohl das illustrative Ausführungsbeispiel der Erfindung, das in den Figuren offenbart ist, einen Lokalspeicher auf der Texturabbildungsplatine aufweist, der als ein Cache implementiert ist, die Erfindung nicht darauf begrenzt ist. Das Texturabbildungssystem kann alternativ derart implementiert sein, daß der Lokalspeicher auf der Texturabbildungsplatine kein Cache ist, wobei andere Techniken verwendet sein können, um sicherzustellen, daß jeder Texturabbildungsdatenblock, der benötigt wird, um ein Grundelement aufzubereiten, durch einen Weg, der von der 3-D-Grundelementpipeline getrennt ist, heruntergeladen wird, bevor das Grundelement aufbereitet wird, so daß die Texturabbildungsdaten von dem Lokalspeicher verfügbar sind, wenn das Grundelement aufbereitet wird.

[0186] Ferner sollte es offensichtlich sein, daß das Schema der vorliegenden Erfindung für das Erzeugen einer Unterbrechung zu einem Rostcomputer,

um Datenblöcke in einem Lokalspeicher zu aktualisieren, mit vielen anderen Anwendungen verwendet werden kann und nicht auf die Verwendung in einem Texturabbildungs-Hardwaresystem begrenzt ist. Dieses Schema ist in jedem Datenverarbeitungssystem brauchbar, das einen Hostcomputer mit einem Hauptspeicher, der Datenblöcke speichert, die verarbeitet werden sollen, und eine Datenverarbeitungshardware, die einen Lokalspeicher aufweist, der Datenblöcke speichert, die verarbeitet werden sollen, aufweist.

X. Cache-Blockersetzungsschema

[0187] sWenn ein Fehlzugriff für einen Texturdatenblock, der sich nicht in dem Cache befindet, auftritt, lädt, wie oben erläutert wurde, der Hostcomputer den angeforderten Texturdatenblock in den Cache **48** (**Fig.** 2). Wenn der Cache voll war, als der Fehlzugriff auftrat, wird einer der Cache-Blöcke durch den neu heruntergeladenen Texturdatenblock ersetzt. Bei einem Ausführungsbeispiel der Erfindung wird eine Bestimmung durchgeführt, dahingehend, welcher Cache-Block vor der längsten Zeit verwendet wurde, wobei dieser Block für die Ersetzung ausgewählt wird, um aktive Blöcke in dem Cache zu halten. Die Bestimmung dessen, welcher Cache-Block zu ersetzen ist, wird durch eine Softwareroutine durchgeführt, die in dem Speicher **17** in dem Hostcomputer (**15**) gespeichert ist, und auf einem Prozessor **19** in dem Hostcomputer ausgeführt wird. Der Texturabbildungschip **46** weist zwei Registersätze auf, die die Softwareroutine beim Bestimmen dessen, welcher Cache-Block ersetzt werden soll, unterstützt. Wenn ein Cache-Fehlzugriff auftritt, werden diese Register durch den 3-D-Umgehungs-Datenweg von dem Hostcomputer gelesen und beim Bestimmen, welcher Cache-Block ersetzt werden soll, verwendet.

[0188] Der erste Registersatz weist zwei Zuletzt-Verwendet-32-Bit-Register MRU0 und MRU1 (zusammen MRU) auf, die jeweils den Bänken null und eins des Cache **48** zugeordnet sind. Jedes Bit dieser Register ist einem der zweiunddreißig Cache-Blöcke, die in der demselben zugeordneten Cache-Bank enthalten sind, zugeordnet. Jedesmal, wenn ein Treffer auf einen Block in dem Cache stattfindet, wird das entsprechende Bit in MRU0 oder MRU1 gesetzt, derart, daß die Zuletzt-Verwendet-Register Treffer für den Cache akkumulieren.

[0189] Der zweite Registersatz weist zwei Gegenwärtig-Verwendet-32-Bit-Register CU0 und CU1 (zusammen CU) auf, die ebenfalls jeweils den Bänken null und eins des Cache zugeordnet sind. Wenn ein Bit entweder in CU0 oder in CU1 gesetzt ist, zeigt dies an, daß der entsprechende Cache-Block gegenwärtig in einem Mini-Verzeichnis des Caches ist und nicht ersetzt werden sollte. Das Cache-Mini-Verzeichnis ist detailliert nachfolgend beschrieben.

[0190] Wenn ein Cache-Fehlzugriff auftritt und den Hostcomputer unterbricht, wird die Softwareroutine, die in dem Flußdiagramm von **Fig.** 18 dargestellt ist, durch den Prozessor **19** des Hostcomputers ausgeführt, um zu bestimmen, welcher Cache-Block durch den ersetzt werden soll, der die angeforderten Texturdaten, die heruntergeladen werden sollen, enthält. Die Softwareroutine hält zwei 64-Bit-Statusworte (d.h. BLÖCKE_-ZUR_VERWENDUNG (BLOCKS_TO_USE) und BLÖCKE BELEGT (BLOCKS_-BUSY), die beim Implementieren der Ersetzungsroutine verwendet werden. Jedes der vierundsechzig Statusbits in diesen Statusworten ist einem der vierundsechzig Cache-Blöcke zugeordnet.

[0191] Wie in einem Schritt **300** gezeigt ist, wird BLÖCKE_ZUR_VERWENDUNG derart initialisiert, daß jedes seiner Bits aktiviert ist, was anzeigt, daß jedes anfänglich für eine Ersetzung verfügbar ist. In einem Schritt **302** überprüft das Verfahren kontinuierlich, um zu bestimmen, ob eine Cache-Fehlzugriff-Unterbrechung empfangen wurde, wobei, wenn eine solche erfaßt wird, das Verfahren zu einem Schritt **304** springt. Im Schritt **304** liest das Verfahren die Register MRU und CU durch den 3-D-Umgehungsdatenweg. Wie oben erläutert wurde, halten bei dem Ausführungsbeispiel der Erfindung, bei dem zwei Texturabbildungschips verwendet sind, die Cache-Speicher in den zwei Chips zu allen Zeiten die gleichen Texturdaten. Wenn das System zwei Texturabbildungschips **46** aufweist, werden folglich die Register MRU und CU von beiden gelesen, derart, daß das Verfahren einen Cache-Block zur Ersetzung auswählen kann, der vor der längsten Zeit in beiden Texturabbildungschips verwendet wurde. In einem Schritt **306** deaktiviert das Verfahren die Bits in BLÖCKE_ZUR_VERWENDUNG, die den Bits entsprechen, die entweder in MRU oder CU aktiviert sind. Bei diesem Ausführungsbeispiel, bei dem zwei oder mehr Texturabbildungschips verwendet sind, wird eine logische ODER-Verknüpfung der MRUs und CUs verwendet, um zu bestimmen, welche Bits in BLÖCKE_ZUR_VERWENDUNG deaktiviert sind.

[0192] In einem Schritt **308** wird eine Bestimmung durchgeführt, dahingehend, ob irgendwelche Bits in BLÖCKE_ZUR_VERWENDUNG aktiviert sind, wobei, wenn zumindest eines aktiviert ist, das Verfahren zu einem Schritt **310** springt, in dem eine Bestimmung durchgeführt wird, dahingehend, ob die Anzahl der aktivierten Bits in BLÖCKE_ZUR_VERWENDUNG unter einer vorbestimmten Schwelle ist. Dieser Schritt wird durchgeführt, um das Beibehalten einer Geschichte der Cache-Blockverwendung über mehrere Cache-Fehlzugriffe zu unterstützen, und um eine ordnungsgemäße Handhabung zukünftiger Cache-Fehlzugriff-Unterbrechungen auf die nachfolgend erläuterte Art und Weise sicherzustellen. Wenn die Anzahl von aktivierten Bits in BLÖCKE BELEGT unter der vorbestimm-

ten Schwelle ist, springt das Verfahren zu einem Schritt **312**, in dem alle Bits in den MRUs deaktiviert werden. Folglich werden die MRUs beginnen, Treffer in dem Cache zu akkumulieren, die nur nach dem Cache-Fehlzugriff auftreten, der gegenwärtig durch das Verfahren verarbeitet wird. Bei einem Ausführungsbeispiel der Erfindung ist die Schwelle bei elf Bits, die in BLÖCKE_ZUR_-VERWENDUNG aktiviert sind, eingerichtet, was anzeigt, daß elf Cache-Blöcke für eine Ersetzung verfügbar sind.

[0193] Nachdem die MRUs im Schritt **312** gelöscht sind, oder wenn im Schritt **310** bestimmt wird, daß die Anzahl von aktivierten Bits in BLÖCKE_ZUR_VERWENDUNG nicht unter die vorbestimmte Schwelle gefallen ist, springt das Verfahren zu einem Schritt **314**, bei dem eines der Bits, die in BLÖCKE_ZUR_VERWENDUNG aktiviert sind, für eine Ersetzung durch den neuen Texturdatenblock, der heruntergeladen werden soll, ausgewählt wird. Der Block, der im Schritt **314** für die Ersetzung ausgewählt wird, wird auf eine Art und Weise; die nachfolgend in Verbindung mit dem Verfahren von **Fig.** 20 erläutert wird, durch den neuen Texturdatenblock ersetzt. Nachdem der Block, der ersetzt werden soll, im Schritt **314** ausgewählt ist, springt das Verfahren zum Schritt **302** zurück, um auf eine weitere Cache-Zugriffs-Unterbrechung zu warten.

[0194] Wenn im Schritt **308** bestimmt wird, daß keine Bits in BLÖCKE_ZUR_VERWENDUNG aktiviert sind, springt das Verfahren zu einem Schritt **316**, in dem BLÖCKE_BELEGT gleich einer logischen ODER-Verknüpfung der MRUs und CUs gesetzt wird. Folglich entsprechen die einzigen Bits, die in BLÖCKE_BELEGT aktiviert sind, denjenigen, die in einem der MRU- oder CU-Register aktiviert sind. Danach wird BLÖCKE_ZUR_VERWENDUNG gleich dem Komplement von BLÖCKE_BELEGT eingestellt. Auf diese Weise wird jedes Bit in BLÖCKE_ZUR_VERWENDUNG aktiviert, mit Ausnahme derjenigen, die den Bits entsprechen, die in den MRUs und CUs aktiviert sind, was anzeigt, daß diese Blöcke nicht für eine Ersetzung ausgewählt werden sollen.

[0195] Nachdem BLÖCKE_ZUR_VERWENDUNG im Schritt **316** gleich dem Komplement von BLÖCKE_BELEGT eingestellt wurde, springt das Verfahren zu einem Schritt **318**, in dem eine Bestimmung durchgeführt wird, dahingehend, ob irgendwelche Bits in BLÖCKE_ZUR_VERWENDUNG aktiviert sind. Wenn zumindest ein Bit in BLÖCKE_ZUR_VERWENDUNG aktiviert ist, springt das Verfahren zu den Schritten **310** bis **314**, in denen die MRUs gelöscht werden, wenn die Anzahl der aktivierten Bits in BLÖCKE_ZUR_VERWENDUNG unter die Löschschwelle gefallen ist, und eines der aktivierten Bits in BLÖCKE_ZUR_VERWENDUNG wird auf die oben beschriebene Art und Weise für die Ersetzung ausgewählt.

[0196] Wenn im Schritt **318** bestimmt wird, daß keine Bits in BLÖCKE_ZUR_VERWENDUNG aktiviert sind, springt das Verfahren zu einem Schritt **320**, in dem drei Aktionen durchgeführt werden. Zuerst werden die MRUs gelöscht, da die Anzahl von Bits, die in BLÖCKE_ZUR_VERWENDUNG aktiviert sind, notwendigerweise unter die vorbestimmte Schwelle gefallen ist. Zweitens wird BLÖCKE_BELEGT gleich den CU-Registern eingestellt. Wie oben erwähnt wurde, zeigt jedes CU-Register die Cache-Blöcke an, die gegenwärtig in dem entsprechenden Cache-Mini-Verzeichnis desselben gehalten sind und daher nicht ersetzt werden sollen. Wenn zwei oder mehr Texturabbildungschips verwendet sind, wird BLÖCKE_BELEGT gleich der logischen ODER-Verknüpfung der CU-Register eingestellt. Schließlich wird BLÖCKE_ZUR_VERWENDUNG gleich dem Komplement von BLÖCKE_BELEGT eingestellt. Folglich ist jedes Bit von BLÖCKE_ZUR_VERWENDUNG aktiviert, mit Ausnahme derjenigen, die den Datenblöcken entsprechen, die gegenwärtig in dem Cache-Mini-Verzeichnis von einem der Texturabbildungschips gehalten sind. Das Verfahren springt dann zu dem Schritt **314**, in dem eines der aktivierten Bits in BLÖCKE_ZUR_VERWENDUNG für die Ersetzung ausgewählt wird. Auf diese Weise können alle Blöcke in dem Cache mit Ausnahme derjenigen in dem Mini-Verzeichnis für eine Ersetzung ausgewählt werden.

[0197] Das Ausführungsbeispiel der vorliegenden Erfindung, das in **Fig.** 18 gezeigt ist, verwendet ein Ersetzungsschema, das einen vor der längsten Zeit verwendeten Cache-Block ersetzt, wenn ein Cache-Fehlzugriff auftritt. Es sollte offensichtlich sein, daß verschiedene Modifikationen bezüglich dieses Schemas durchgeführt werden können, ohne von dem Bereich der vorliegenden Erfindung abzuweichen. Beispielsweise ist bei dem Ausführungsbeispiel, das in **Fig.** 18 gezeigt ist, das MRU-Hardwareregister verwendet, um Treffer in dem Cache über eine Zeitperiode zu sammeln, die potentiell mehrere Cache-Fehlzugriffe einschließen kann, wobei das MRU-Register nur gelöscht wird, sobald die Anzahl von Bits, die in BLÖCKE_-ZUR_VERWENDUNG aktiviert sind, unter die vorbestimmte Schwelle gefallen ist. Außerdem wird das Softwarestatuswort BLÖCKE_BELEGT in den Schritten **316** oder **320** nur aktualisiert, wenn bestimmt wird, daß keine Bits in BLÖCKE_ZUR_VERWENDUNG aktiviert sind. Das Ersetzungsschema kann alternativ durch das Aktualisieren von BLÖCKE_BELEGT von dem MRU-Register jedesmal, wenn eine Cache-Fehlzugriff-Unterbrechung empfangen wird, und dann das Löschen des MRU-Registers implementiert sein. Auf diese Weise kann das Softwarestatuswort BLÖCKE_BELEGT verwendet sein, um die Ge-

schichte der Treffer in dem Cache über eine Zeitperiode zu akkumulieren, die potentiell mehrere Cache-Fehlzugriffe einschließen kann, und das Hardwareregister MRU kann verwendet sein, um nur Treffer zwischen Fehlzugriffen zu akkumulieren.

[0198] Obwohl die Schwelle der aktivierten Bits in BLÖCKE_ZUR_VERWENDUNG, die das Löschen der MRUs zur Folge hat, bei dem veranschaulichenden oben beschriebenen Ausführungsbeispiel auf elf verfügbare Blöcke eingestellt ist, sollte es ferner offensichtlich sein, daß diese Anzahl naheliegend geändert werden kann. Diese Schwelle beeinflußt die Anzahl von Malen, die die Routine im Schritt **308** eine Situation antreffen wird, bei der keines der Bits in BLÖCKE_ZUR_VERWENDUNG aktiviert ist. Es ist erwünscht, diese Situation zu vermeiden, da dieselbe das Aktualisieren von BLÖCKE_ZUR_VERWENDUNG (im Schritt **316** oder **320**) mit nur der jüngsten Geschichte der Cache-Blockverwendung zur Folge hat, d.h. der Geschichte nach dem vorher verarbeiteten Cache-Fehlzugriff. Es ist bevorzugt, einen höheren Auflösungsgrad zu schaffen, derart, daß die Bits, die in BLÖCKE_ZUR_VERWENDUNG aktiviert sind, Blöcke wiederspiegeln, die während der Verarbeitung von mehreren Cache-Fehlzugriffen nicht verwendet wurden, wenn derartige Blöcke existieren. Folglich kann durch das Steuern der Schwelle der aktivierten Bits in BLÖCKE_ZUR_VERWENDUNG, die das Löschen der MRUs zur Folge haben, die Anzahl von Durchläufen durch das Verfahren, bei denen keines der Bits von BLÖCKE_ZUR_VERWENDUNG im Schritt **308** aktiviert wird, minimiert werden, wobei ein gewünschter Auflösungspegel beim Bestimmen eines vor der längsten Zeit verwendeten Cache-Blocks geschaffen wird.

[0199] Es sollte offensichtlich sein, daß das oben beschriebene Blockersetzungsschema, das durch eine Softwareroutine implementiert ist, die auf einem Hostcomputer ausgeführt wird, nicht auf die Verwendung mit einem Cache-Speicher begrenzt ist. Diese Ersetzungsroutine kann in jedem Datenverarbeitungssystem verwendet werden, bei dem ein Lokalspeicher Blöcke von Daten, die verarbeitet werden, aufweist, und bei dem, wenn zusätzliche Datenblöcke von einem Hostcomputer in den Lokalspeicher heruntergeladen werden, Datenblöcke in dem Lokalspeicher ersetzt werden.

XI. Ausschalten der Cache-Operation

[0200] Bei einem Ausführungsbeispiel der Erfindung ist die Fähigkeit vorgesehen, die Cache-Operation des Lokalspeichers **48** auf der Texturabbildungsplatine durch das Ausschalten von Cache-Fehlzugriffen auszuschalten, derart, daß Texturdaten für jedes 3-D-Grundelement in den Speicher **48** heruntergeladen werden, bevor sie während der Aufbereitung des Grundelements erforderlich sind. Jeder Texturabbildungschip **46** weist ein Statusbit auf, das anzeigt, daß die Operation seines Lokalspeichers als ein Cache freigegeben ist. Wenn dieses Statusbit aktiviert ist, haben Cache-Fehlzugriffe eine Unterbrechung des Hostcomputers und ein Anhalten des Texturabbildungschips zur Folge. Wenn das Statusbit jedoch deaktiviert ist, arbeitet der Lokalspeicher **48** auf der Texturabbildungsplatine nicht als ein Cache und die Texturdaten für jedes Grundelement werden in den Speicher **48** heruntergeladen, bevor sie durch das Grundelement benötigt werden, so daß Fehlzugriffe auf den Speicher nicht stattfinden. Bei einem Ausführungsbeispiel der Erfindung werden, wenn die Operation des Lokalspeichers als ein Cache ausgeschaltet ist, Texturdaten durch die 3-D-Grundelementpipeline in den Lokalspeicher auf der Texturabbildungsplatine heruntergeladen, um eine Synchronisation der Texturdaten mit den entsprechenden 3-D-Grundelementdaten zu erleichtern.

XII. Texeltorregister, die das Schema zum Herunterladen von Texturdaten als Reaktion auf einen Cache-Fehlzugriff unterstützen

[0201] Wie oben erläutert wurde, weist der Texturabbildungschip **46** (**Fig.** 2) einen Texeltor **92** (**Fig.** 3) auf, das verwendet ist, um Texturdaten zu empfangen, die von dem Rostcomputer **15** heruntergeladen werden. Das Texeltor weist eine Anzahl von Registern auf, die das Herunterladen der Texturdaten unterstützen. Einige dieser Register wurden oben erläutert, einschließlich der Register MRU und CU. Die anderen Texeltorregister schließen ein Befehlsregister, ein Statusregister, ein Texeldatenregister, ein Verzeichniskennungsregister, ein Cache-Adreßregister und ein Pipelinekennungsregister ein, die jeweils Funktionen durchführen, die nachfolgend erläutert werden.

[0202] Ein Zugriff auf die Texeltorregister ist vorgesehen, um zu ermöglichen, daß dieselben durch die 3-D-Grundelementpipeline beschrieben werden. Die Texeltorregister können beschrieben werden, selbst wenn die 3-D-Pipeline belegt ist, wobei die Daten zum Beschreiben der Register einfach in der Pipeline plaziert werden. Außerdem kann durch die 3-D-Pipelineumgehung, die über den 24-Bit-Bus **24** (**Fig.** 2) vorgesehen ist, auf die Texeltorregister zugegriffen werden. Wenn auf die Texeltorregister zugegriffen wird, werden acht Bits des Busses **24** als eine Registeradresse verwendet, um zu spezifizieren, welches Texeltorregister gelesen oder beschrieben werden soll, wobei, wenn Daten in ein Texeltorregister geschrieben werden, die anderen sechzehn Bits des Busses die Daten liefern.

[0203] Die Organisationen der Texeltorregister sind in **Fig.** 19 gezeigt. Bei einem Ausführungsbeispiel der Erfindung weist jedes der Texeltorregister zwei-

unddreißig Bit auf, obwohl eine Anzahl der Bits in einigen der Register nicht verwendet ist.

A. Texelbefehlsregister

[0204] Das Texelbefehlsregister weist eine Anzahl von Bits auf, die durch die Hostcomputer-Softwareroutine verwendet werden, wie detaillierter nachfolgend erläutert wird, die Cache-Fehlzugriffe abwickelt. Ein Halt-Bit **350** wird durch die Softwareunterbrechungs-Handhabungsroutine gesetzt und weist den Texturabbildungschip an, seinen Betrieb anzuhalten. Wie oben erwähnt wurde, werden bei dem Ausführungsbeispiel der Erfindung, bei dem zwei Texturabbildungschips vorgesehen sind, beide Texturabbildungschips als Reaktion auf einen Cache-Fehlzugriff von einem mit den gleichen Texturdaten aktualisiert, so daß die Caches konsistent bleiben. Wenn folglich ein Fehlzugriff von einem Texturabbildungschip empfangen wird, werden beide durch das Setzen des Halt-Bits **350** in ihren jeweiligen Texelbefehlsregistern angehalten. Das Halt-Bit wird durch die Softwareroutine, die den Cache-Fehlzugriff handhabt, gelöscht, indem in das Befehlsregister geschrieben wird, das Bit zu löschen, nachdem neue Texturdaten als Reaktion auf den Cache-Fehlzugriff von vom Hostcomputer heruntergeladen wurden.

[0205] Ein Unterbrechung-Freigegeben-Bit **352** gibt, wenn es aktiviert ist, Unterbrechungen von dem Texeltor frei, wenn ein Cache-Fehlzugriff stattfindet. Dieses Bit wird deaktiviert, um die oben beschriebene Fähigkeit zu liefern, daß der Lokalspeicher **48** auf der Texturabbildungsplatine **12** (**Fig.** 2) nicht als ein Cache arbeitet.

[0206] Schreib-Loki0- und Schreib-Lokil-Bits **354** und **356** sind Schreibfreigaben für die Texeltorregister. Loki ist eine Abkürzung, die verwendet ist, um den Texturabbildungschip **46** zu identifizieren. Bei dem Ausführungsbeispiel der Erfindung, bei dem zwei solche Chips verwendet sind, werden die Chips jeweils als Loki0 und Loki1 bezeichnet. Wenn nur ein einzelner Texturabbildungschip verwendet ist, wird derselbe als Loki0 identifiziert. Wenn ein Befehl über den Texeltorbus **24** empfangen wird, in ein beliebiges der Texeltorregister zu schreiben, überprüft jeder Texturabbildungschip (d.h. Loki0 und Loki1) sein Befehlsregister, um zu bestimmen, ob sein Schreibbit freigegeben ist, und aktualisiert, wenn dies der Fall ist, seine Texeltorregister gemäß dem empfangenen Schreibbefehl. Folglich kann eine Softwareroutine, die auf dem Hostcomputer läuft, durch das Steuern der Werte der Schreib-Loki0- und Schreib-Loki1-Bits **354** und **356** entweder getrennt oder kombiniert in die Texeltorregister in den zwei Texturabbildungschips schreiben.

[0207] Ein Loki-Lesebit **358** gibt ein Lesen der Texeltorregister von einem der Texturabbildungschips frei. Wenn ein Befehl über den Texelbus **24** empfangen wird, ein Texeltorregister zu lesen, spricht nur jeweils einer der Texturabbildungschips an, um den Inhalt seines Texeltorregisters auf dem Bus zu liefern. Bei dem Ausführungsbeispiel, bei dem zwei Texturabbildungschips vorgesehen sind, kann jeder mit einem Anschlußstift versehen sein, der fest verdrahtet ist, um anzuzeigen, ob der Chip Loki0 oder Loki1 ist. Wenn das Loki-Lesebit durch die Software gesetzt ist, zeigt es an, daß Lesevorgänge von Loki1 freigegeben sind, wobei, wenn das Lesebit deaktiviert ist, dasselbe anzeigt, daß Lesevorgänge für Loki0 freigegeben sind. Aus dem vorhergehenden sollte offensichtlich sein, daß es das Format des Texelbefehlsregisters ermöglicht, daß dasselbe für beide Texturabbildungschips (Loki0 und Loki1) gleichzeitig mit denselben Daten beschrieben wird, wodurch nur ein einziger Schreibzyklus erforderlich ist, um beide Befehlsregister zu beschreiben.

B. Texelstatusregister

[0208] Das Texeltor-Statusregister weist ein Dual-Loki-Bit **360** auf, das, wenn es aktiviert ist, anzeigt, daß das System zwei Texturabbildungschips aufweist. Ein Unterbrechung-Freigegeben-Bit **362** ist aktiviert, wann immer das Bit **352** in dem Befehlsregister aktiviert ist, und zeigt an, daß der Lokalspeicher in dem Texturabbildungschip als ein Cache arbeitet, der Fehlzugriffe erzeugen wird, die den Hostcomputer unterbrechen, wenn Texturdaten benötigt werden, die sich nicht in dem Cache befinden. Dieses Bit befindet sich sowohl in dem Statusregister als auch in dem Befehlsregister, derart, daß der Status des Texeltors einfach durch das Lesen des Statusregisters gelesen werden kann.

[0209] Ein Unterbrechung-Gültig-Bit **364** ist aktiviert, wenn eine Unterbrechung von dem Texturabbildungschip aufgetreten ist, und der Chip auf neue Texturdaten, die heruntergeladen werden sollen, wartet. Dieses Bit wird gelöscht, wenn das Cache-Verzeichnis-Kennungsregister (das nachfolgend erläutert wird), mit einer Cache-Kennung beschrieben wird, die mit der Cache-Lesekennung, die in dem Pipelinekennungsregister (das nachfolgend erläutert wird) gespeichert ist, welches die Kennung ist, die einen Fehlzugriff in dem Cache bewirkte, übereinstimmt.

[0210] Das Statusregister weist zwei Bits auf, die das Anhalten des Texturabbildungschips unterstützen, wenn ein Cache-Fehlzugriff stattfindet. Ein Anhalten-Freigegeben-Bit **368** wird durch die Softwareroutine, die auf dem Hostcomputer läuft, gesetzt und gelöscht, wann immer das Halt-Bit **350** in dem Befehlsregister jeweils gesetzt und gelöscht wird, und weist den Texturabbildungschip an, sich selbst anzuhalten, wenn das Bit aktiviert ist. Dieses Bit ist sowohl in dem Statusregister als auch dem Befehlsregister vorgesehen, so daß der Status des Texturabbil-

dungschips in einem einzelnen Register gespeichert ist. Das Unterbrechung-Gültig-Bit **364** wird durch die Hardware in dem Texturabbildungschip gesetzt, wenn ein Cache-Fehlzugriff aufgetreten ist und das Cache-Verzeichnis auf Daten, die heruntergeladen werden sollen, wartet. Dieses Bit wird gelöscht, wenn das Cache-Verzeichniskennungsregister (das nachfolgend erläutert wird) mit einer Cache-Kennung beschrieben wird, die mit der Blockkennung, die den Fehlzugriff in dem Cache bewirkte, übereinstimmt.

C. Pipelinekennungsregister

[0211] Das Pipelinekennungsregister speichert die letzte Blockkennung, die durch die Pipeline in dem Texturabbildungschip indiziert wurde. Wenn ein Cache-Fehlzugriff stattfindet, speichert das Pipelinekennungsregister die Blockkennung **370**, die den Fehlzugriff in dem Cache bewirkte. Folglich kann die Software, die auf die Cache-Fehlzugriff-Unterbrechung anspricht, durch das Lesen des Pipelinekennungsregisters über den Texeltorbus **24** die Kennung für den Cache-Block bestimmen, der als Reaktion auf den Fehlzugriff in den Cache heruntergeladen werden soll.

D. Texeldatenregister

[0212] Das Texeldatenregister wird verwendet, um Texturdaten in den Cache **48** herunterzuladen, wenn ein Cache-Fehlzugriff stattfindet. Wie oben erwähnt wurde, ist jedes Texel durch zweiunddreißig Datenbits dargestellt, wobei ein Byte **372** Alpha darstellt, ein Byte **374** den Rot-Wert darstellt, ein Byte **376** den Grün-Wert darstellt und ein Byte **378** den Blau-Wert darstellt.

E. Texel-Cache-Adreßregister

[0213] Das Texel-Cache-Adreßregister ist verwendet, um Texeldaten in den Cache und Blockkennungen in das Cache-Verzeichnis zu schreiben. Wie oben erläutert wurde, speichert der Cache vierundsechzig Texturdatenblöcke, wobei jeder Block ein Array von 256 × 256 Texeln aufweist. Das Texel-Cache-Adreßregister weist ein 6-Bit-Blockindexfeld **380** auf, das den Speziellen der vierundsechzig Blöcke in dem Cache, aus dem gelesen oder in den geschrieben werden soll, identifiziert. Zusätzlich weist das Register ein 16-Bit-Blockadreßfeld **382** auf, das die spezielle Texeladresse identifiziert, die in dem Block, der in dem Blockindexfeld identifiziert ist, gelesen oder geschrieben wird. Wenn Daten als Reaktion auf einen Cache-Fehlzugriff in den Texturspeicher heruntergeladen werden, wird der Blockindex durch die Softwareroutine unter Verwendung des Vor-Der-Längsten-Zeit-Verwendet-Ersetzungsschemas, das oben erläutert wurde, eingestellt, und das Blockadreßfeld **382** wird auf Nullen initialisiert, um das erste Texel in den Block zu schreiben. Das Cache-Adreßregister inkrementiert das Blockadreßfeld **382** automatisch, immer wenn auf das Texeldatenregister zugegriffen wird. Folglich kann das Blockadreßfeld durch alle Blockadressen in dem Cache-Block inkrementiert werden, um den neuen Texeldatenblock in den Cache zu schreiben.

F. Texelverzeichniskennungsregister

[0214] Das Texelverzeichniskennungsregister weist ein 23-Bit-Blockkennungsfeld **384** auf, das die Cache-Blockkennung darstellt und verwendet ist, um den Cache-Verzeichniseintrag, der durch das Blockindexfeld **380** in dem Cache-Adreßregister definiert ist, zu schreiben. Wie oben erläutert wurde, stellen die dreiundzwanzig Bits der Cache-Blockkennung acht Bits der Textur-ID, sieben Bits der S-Koordinaten, sieben Bits der T-Koordinaten und ein zusätzliches Bit dar, das die Tabellennummer der Tabelle in der Reihe von MIP-Tabellen, die durch den Texturdatenblock, der der Blockkennung entspricht, dargestellt ist, identifiziert. Wenn ein neuer Texturdatenblock als Reaktion auf einen Cache-Fehlzugriff von dem Rostcomputer heruntergeladen wird, wird seine Blockkennung über den Texelbus **24** in das Verzeichniskennungsregister geladen. Aus dem Verzeichniskennungsregister wird die Blockkennung in den Eintrag in dem Cache-Verzeichnis geschrieben, der durch das Blockindexfeld **380** des Cache-Adreßregisters identifiziert ist. Wenn eine Blockkennung in das Verzeichniskennungsregister geschrieben wird, die mit der Kennung in dem Pipelinekennungsregister (die diejenige ist, dessen Lesevorgang einen Cache-Fehlzugriff zur Folge hatte) übereinstimmt, wird, wie oben erwähnt wurde, die Cache-Fehlzugriff-Unterbrechung gelöscht.

XIII. Softwareroutine zum Abwickeln von Cache-Fehlzugriff-Unterbrechungen

[0215] Wie aus dem vorhergehenden offensichtlich sein sollte, werden die Texeltorregister durch eine Softwareroutine verwendet, die auf dem Hostcomputer **15** läuft, welche Cache-Fehlzugriff-Unterbrechungen abwickelt, um die notwendigen Texturdaten herunterzuladen. Ein Flußdiagramm dieser Softwareroutine ist in **Fig.** 20 gezeigt. In einem Schritt **400** wird das Texelbefehlsregister sowohl für Loki0 als auch für Loki1 beschrieben, um das Halt-Bit **350** in beiden zu setzen. Das Verfahren springt dann zu einem Schritt **402**, um das Angehalten-Bit **368** in den Texelstatusregistern zu lesen, um zu bestimmen, ob beide Lokis angehalten wurden. Das Verfahren liest die Statusregister von Loki0 und Loki1 kontinuierlich, bis bestimmt ist, daß beide angehalten wurden, und springt dann zu einem Schritt **404**. Wenn das System nur einen einzigen Texturabbildungschip **46** (d.h. Loki0) aufweist, spricht Loki0 ebenfalls auf Anforderungen an, um die Texeltorregister von Loki1 zu lesen, indem der Inhalt seiner Register auf dem Texelbus **24** gelie-

fert wird. Wenn die Softwareroutine im Schritt **402** überprüft, um zu bestimmen, ob beide Lokis angehalten wurden, spricht Loki0 folglich auf Lesevorgänge von Loki1 an, derart, daß, wenn Loki0 angehalten wurde, das Verfahren zum Schritt **404** springt.

[0216] Im Schritt **404** wird das Unterbrechung-Gültig-Bit **364** in dem Texelstatusregister von Loki0 gelesen, um zu bestimmen, ob Loki0 unterbrochen hat, um den Cache-Fehlzugriff zu bewirken, wobei, wenn dies der Fall ist, das Verfahren zu einem Schritt **406** springt, in dem das Pipelinekennungsregister von Loki0 gelesen wird, um die Blockkennung des Texturdatenblocks zu lesen, der in dem Cache den Fehlzugriff bewirkte. Die Softwareroutine verwendet diese Blockkennung, um auf den entsprechenden Texturdatenblock in dem Speicher **17** (**Fig.** 2) des Hostcomputers zuzugreifen, und springt zu einem Schritt 408, um zu bestimmen, welcher Block in dem Cache durch den neuen Texturdatenblock, der heruntergeladen werden soll, ersetzt werden soll. Diese Bestimmung wird unter Verwendung des Vor-Der-Längsten-Zeit-Verwendet-Schemas, das oben in Verbindung mit **Fig.** 18 beschrieben ist, durchgeführt.

[0217] Wenn das System zwei Texturabbildungschips aufweist, sind die Caches in jedem eingerichtet, um identische Einträge aufzuweisen, wie oben dargelegt wurde. Daher werden Texturdaten, die als Reaktion auf einen Cache-Fehlzugriff von einem der Texturabbildungschips von dem Rostcomputer heruntergeladen werden, in die Caches in beiden Chips geschrieben. Sobald der Cache-Block, der ersetzt werden soll, identifiziert wurde, springt das Verfahren folglich zu einem Schritt **410**, in dem die Cache-Adreßregister in Loki0 und Loki1 (wenn Loki1 existiert) mit dem Blockindex beschrieben werden, der während des Schritts **408** bestimmt wird. In einem Schritt **412** wird das Verzeichniskennungsregister mit der Blockkennung des Texturdatenblocks, der als Reaktion auf den Cache-Fehlzugriff in den Textur-Cache heruntergeladen werden soll, geschrieben, wobei in einem Schritt **414** die Texturdaten in das Texeldatenregister geschrieben werden. Auf diese Weise spricht das Verfahren durch das Herunterladen des Texturdatenblocks, der den Fehlzugriff in dem Cache bewirkte, und das Schreiben dieses Datenblocks in den Cache auf den Cache-Fehlzugriff an.

[0218] Nachdem der Texturdatenblock in den Schritten **406** bis **414** in Loki0 und Loki1 heruntergeladen wurde, oder wenn im Schritt **404** bestimmt wird, daß Loki0 nicht unterbrochen hat, springt das Verfahren zu einem Schritt **416**, in dem eine Bestimmung durchgeführt wird, dahingehend, ob das Unterbrechung-Gültig-Bit **364** in dem Statusregister von Loki1 gesetzt wurde, was anzeigt, daß ein Cache-Fehlzugriff in Loki1 stattgefunden hat. Wenn das System nur einen einzelnen Texturabbildungschip aufweist, spricht, wie oben erläutert wurde, Loki0 auf Lesevorgänge aus den Texeltorregistern von Loki1 an. Wenn Loki0 auf ein Lesen aus dem Statusregister von Loki1 anspricht, maskiert es sein Unterbrechung-Gültig-Bit **364**, so daß die Softwareroutine im Schritt **416** bestimmen wird, daß Loki1 nicht unterbrochen hat. Diese Maskierung wird derart durchgeführt, daß die Softwareroutine die Unterbrechung von Loki0 durch ein erneutes Herunterladen des Texturdatenblocks, der in den Schritten **406** bis **414** heruntergeladen wurde, nicht erneut verarbeitet. Daher wird bei einem System, bei dem nur ein einzelner Texturabbildungschip vorgesehen ist, das Verfahren im Schritt **416** bestimmen, daß Loki1 nicht unterbrochen hat, und wird zum Schritt **418** springen, in dem das Befehlsregister in Loki0 beschrieben wird, um das Halt-Bit **350** zu deaktivieren, was den Texturabbildungschip frei gibt, um mit dem Verarbeiten der Grundelemente in seiner Pipeline fortzufahren.

[0219] Wenn das System zwei Texturabbildungschips aufweist, wird das Verfahren im Schritt **416** bestimmen, ob Loki1 unterbrochen hat, und wird, wenn dies nicht der Fall ist, ebenfalls direkt zum Schritt **418** springen, in dem das Halt-Bit in beiden Texturabbildungschips deaktiviert wird, was ermöglicht, daß dieselben mit dem Verarbeiten der Grundelemente fortfahren. Wenn im Schritt **416** jedoch bestimmt wird, daß Loki1 als Reaktion auf einen Cache-Fehlzugriff unterbrochen hat, fährt das Verfahren mit den Schritten **420** bis **424** fort, um die Unterbrechung auf die gleiche Art und Weise zu verarbeiten, die in Verbindung mit den Schritten **406** bis **414** zum Handhaben der Unterbrechung von Loki0 erläutert wurde. Das Verfahren springt dann zu dem Schritt **418**, in dem die Halt-Bits in beiden Texturabbildungschips deaktiviert werden.

[0220] Es sollte offensichtlich sein, daß bei einem System, bei dem zwei Texturabbildungschips vorgesehen sind, beide Chips gleichzeitig eine Cache-Fehlzugriff-Unterbrechung für die gleiche Blockkennung oder für unterschiedliche Blockkennungen erzeugen können. Wenn beide Texturabbildungschips Cache-Fehlzugriff-Unterbrechungen für die gleiche Blockkennung erzeugen, wird die Unterbrechung in den Schritten **400** bis **414** verarbeitet. Daher wird das Verfahren im Schritt **416** keine Unterbrechung von Loki1 erfassen, da die Unterbrechung von Loki1 durch das Schreiben der Blockkennung, die den Fehlzugriff bewirkte, in das Verzeichniskennungsregister beider Lokis im Schritt **412** gelöscht wird. Folglich ist das Verfahren, das in **Fig.** 20 gezeigt ist, in der Lage, auf eine Unterbrechung von jedem Texturabbildungschip individuell oder von beiden gleichzeitig anzusprechen.

XIV. Cache-Mini-Verzeichnis und -Hauptverzeichnis

[0221] Wie oben dargelegt wurde, weist der Cache

bei einem Ausführungsbeispiel der Erfindung vierundsechzig Blöcke aus 256 × 256 Datentexeln auf und ein voll assoziatives Cache-Verzeichnis, das vierundsechzig Einträge aus 23-Bit-Blockkennungen aufweist. Wenn die vorliegende Erfindung im trilinearen Interpolationsmodus arbeitet, werden acht Texellesevorgänge durchgeführt, um die resultierenden Texeldaten für ein Pixel zu bestimmen, wobei vier Texel in einer Tabelle gleichzeitig in einer Leseoperation gelesen werden, und vier Texel in der anderen Tabelle gleichzeitig in einer zweiten Leseoperation gelesen werden. Wenn die Pixel, auf denen gearbeitet wird, auf einen Ort in einer Tabelle abbilden, der benachbart zu einer Cache-Blockgrenze ist, können die vier Texel, die aus dem Cache gelesen werden, um die resultierenden Texeldaten zu erzeugen, innerhalb einer Tabelle jeweils in einem unterschiedlichen Cache-Block sein. Folglich könnte das gleichzeitige Lesen von vier Texeln aus dem Cache für jedes Pixel vier separate Vergleiche mit den vierundsechzig Blockkennungseinträgen in dem Cache-Verzeichnis erfordern.

[0222] Herkömmliche voll assoziative Caches arbeiten auf eine von zwei Arten. Erstens liefern einige separate Hardwarekomparatoren für jeden Cache-Kennungseintrag, so daß eine Lesekennung in einem einzelnen Zyklus mit jedem Cache-Kennungseintrag verglichen werden kann. Eine solche Technik würde einen großen Hardwareaufwand bei der vorliegenden Erfindung einführen, bei der gleichzeitig vier Lesevorgänge durchgeführt werden, und würde zweihundertsechsundfünfzig (d.h. 4 × 64) 23-Bit-Komparatoren erfordern. Eine zweite Technik, die von herkömmlichen voll assoziativen Caches verwendet wird, verwendet einen einzelnen Cache-Kennungskomparator, wobei jeder Cache-Eintrag seriell mit der Lesekennung verglichen wird. Eine solche Technik würde bei der vorliegenden Erfindung, bei der potentiell zweihundertsechsundfünfzig Lesezyklen von dem Cache-Verzeichnis gefordert werden, um zu bestimmen, ob jedes der vier Texel, das während einer einzelnen Leseoperation gelesen wird, in dem Cache vorliegt, die Systembandbreite negativ beeinflussen.

[0223] Um diese Probleme zu lösen, weist das Cache-System der vorliegenden Erfindung sowohl ein Mini-Verzeichnis (**Fig.** 21) als auch ein Hauptverzeichnis (**Fig.** 22) auf. Das Mini-Verzeichnis ist voll assoziativ und weist die fünf zuletzt gelesenen Cache-Blockkennungen ebenso wie einen entsprechenden Blockindex für jede auf. Wie in **Fig.** 21 gezeigt ist, weist das Mini-Verzeichnis **500** fünf Einträge auf, die jeweils über Ausgänge **501** bis **505**, von denen jeder mit vier Gruppen von Kennungskomparatoren **507** bis **510** gekoppelt ist, von dem Mini-Verzeichnis ausgegeben werden. Jede Gruppe von Kennungskomparatoren **507** bis **510** weist fünf 23-Bit-Komparatoren (nicht gezeigt) auf, und ist einer der vier Cache-Lesekennungen zugeordnet, die bei einer einzelnen Leseoperation durchgeführt werden, wenn eine bilineare oder trilineare Interpolation durchgeführt wird. Folglich ist die voll assoziative Beschaffenheit des Mini-Verzeichnisses mit 20 23-Bit-Komparatoren, was gleich der Anzahl von Kennungen, die gleichzeitig gelesen werden, multipliziert mit der Anzahl der Einträge in das Mini-Verzeichnis ist, implementiert.

[0224] Die vier Cache-Lesekennungen, die für ein Pixel gleichzeitig gelesen werden, identifizieren die Cache-Blöcke, die die vier Texel aufweisen, die am nächsten an dem Ort in der Tabelle sind, auf den das Pixel abbildet, und die als eine obere linke (UL) Kennung, eine obere rechte (UR) Kennung, eine untere linke (LL) Kennung und eine untere rechte (LR) Kennung bezeichnet werden. Die Cache-Lesekennungen für das obere linke, das obere rechte, das untere linke und das untere rechte Texel sind jeweils mit Gruppen von oberen linken, oberen rechten, unteren linken und unteren rechten Kennungskomparatoren **507** bis **510** verbunden. Jede Gruppe von Kennungskomparatoren **507** bis **510** vergleicht seine zugeordnete Cache-Lesekennung gegenüber den fünf Blockkennungen, die in dem Mini-Verzeichnis gespeichert sind und erzeugt eine Trefferausgabe, die anzeigt, ob die Kennung mit einem der Mini-Verzeichniseinträge übereinstimmt, und gibt, wenn dies der Fall ist, ferner einen Blockindex aus, der den Ort in dem Cache anzeigt, an dem der entsprechende Texeldatenblock gespeichert ist.

[0225] Wie aus dem vorhergehenden offensichtlich sein sollte, ist, wenn jede der vier Cache-Lesekennungen (UL, UR, LL, LR) in dem Mini-Verzeichnis ist, nur ein einzelner Verzeichniszugriff erforderlich, um die Blockindizes zu bestimmen, die die Orte in dem Cache identifizieren, an denen die entsprechenden vier Blöcke von Texeldaten gespeichert sind. Ein Zugriff auf das Cache-Hauptverzeichnis wird nur durchgeführt, wenn eine oder mehrere der Lesekennungen nicht in dem Mini-Verzeichnis sind. Das Mini-Verzeichnis **500** wird jedesmal aktualisiert, wenn eine Cache-Lesekennung einen Fehlzugriff in dem Mini-Verzeichnis bewirkt, so daß das Mini-Verzeichnis **500** zu jeder Zeit die Blockkennungen der fünf Texturdatenblöcke aufweist, auf die zuletzt zugegriffen wurde.

[0226] Wenn eine oder mehrere der vier Cache-Lesekennungen in dem Mini-Verzeichnis nicht trifft, wird ein Zugriff auf das Cache-Hauptverzeichnis **520** (**Fig.** 22) durchgeführt. Wie oben dargelegt wurde, weist das Hauptverzeichnis vierundsechzig Einträge auf, von denen jeder eine Blockkennung aufweist. Das Hauptverzeichnis ist mit vierundsechzig 23-Bit-Komparatoren 522 versehen, so daß eine Cache-Lesekennung in einem einzelnen Zyklus mit dem gesamten Hauptverzeichnis verglichen werden kann. Die Komparatoren **522** liefern ein Signal, das anzeigt,

ob die Cache-Lesekennung einen der Einträge in dem Hauptverzeichnis getroffen hat, wobei, wenn dies der Fall ist, der Ort des Komparators, der mit der Lesekennung übereinstimmt, ebenfalls verwendet wird, um einen Blockindex zu erzeugen, der identifiziert, wo sich der entsprechende Texeldatenblock in dem Cache befindet. Wenn die Lesekennung nicht mit irgendeinem der Einträge in dem Cache-Hauptverzeichnis übereinstimmt, wird ein Cache-Fehlzugriff erzeugt, der bewirkt, daß der Hostcomputer unterbrochen wird, um den angeforderten Texturdatenblock auf die Art und Weise, die oben beschrieben wurde, herunterzuladen.

[0227] Wie oben dargelegt wurde, wird nur auf das Cache-Hauptverzeichnis **520** zugegriffen, wenn eine oder mehrere der vier Cache-Lesekennungen (UL, UR, LL, LR) das Mini-Verzeichnis nicht treffen. Wenn zwei oder mehr der Cache-Lesekennungen das Mini-Verzeichnis verfehlen, ist es erwünscht, den Verhaltensnachteil zu reduzieren, der erlitten werden würde, wenn in getrennten Zyklen für jede Cache-Lesekennung auf das Hauptverzeichnis zugegriffen werden müßte. Um dieses Ergebnis zu erreichen, ist bei einem Ausführungsbeispiel der Erfindung eine Gruppe von sechs zusätzlichen Komparatoren **526** bis **530** vorgesehen, wie in **Fig.** 23 gezeigt ist. Die sechs Komparatoren vergleichen jede der vier Cache-Lesekennungen, auf die gleichzeitig zugegriffen wird, mit den anderen, um zu bestimmen, ob irgendwelche identisch sind. Die Komparatoren weisen einen Komparator **526** auf, der die UL-Kennung mit der UR-Kennung vergleicht, einen Komparator **527**, der die UL- und die LL-Kennung vergleicht, einen Komparator **528**, der die UL- und die LR-Kennung vergleicht, einen Komparator **529**, der die UR- und die LL-Kennung vergleicht, einen Komparator **530**, der die UR- und die LR-Kennung vergleicht, und einen Komparator **532**, der die LL- und die LR-Kennung vergleicht.

[0228] Die Vergleiche, die durch die Komparatoren **526** bis **532** durchgeführt werden, können parallel mit anderen Vergleichen durchgeführt werden, um nicht irgendeinen Verhaltensnachteil zu erleiden. Beispielsweise können diese Vergleiche während des Zyklusses durchgeführt werden, zu dem die Cache-Lesekennungen mit dem Mini-Verzeichnis verglichen werden, oder während des Zyklusses, zu dem eine erste Cache-Lesekennung, die einen Fehlzugriff in dem Mini-Verzeichnis bewirkte, mit dem Hauptverzeichnis verglichen wird. Wenn bestimmt wird, daß zumindest zwei Cache-Lesekennungen das Mini-Verzeichnis nicht treffen und gleich sind, werden die Ausgaben der Komparatoren **526** bis **532** verwendet, um anzuzeigen, daß für diese zumindest zwei Cache-Lesekennungen nur einmal auf das Hauptverzeichnis zugegriffen werden muß. Auf diese Art und Weise müssen nicht mehrere Zyklen unter dem Zugreifen auf das Hauptverzeichnis für Kennungen, die identisch sind, leiden, wodurch die Beeinflussung der Systembandbreite minimiert wird, wenn zwei oder mehr Cache-Lesekennungen das Mini-Verzeichnis verfehlen.

[0229] Aus dem vorhergehenden sollte offensichtlich sein, daß das Ausführungsbeispiel der vorliegenden Erfindung, das das Cache-Mini-Verzeichnis verwendet, die sich entgegenstehenden Ziele des Verwendens eines relativ geringen Betrags an Hardware, um das Cache-Verzeichnis zu implementieren, während eine hohe Systembandbreite erreicht wird, ins Gleichgewicht bringt. Die Verhaltensnachteile, die eingeführt werden, wenn zwei oder mehr Cache-Lesekennungen das Mini-Verzeichnis verfehlen, sind anwendungsabhängig. Obwohl es möglich ist, daß zwei eindeutige Sätze von vier Cache-Lesekennungen alle zwei Zyklen durch das Mini-Verzeichnis verarbeitet werden können, wird angenommen, daß typischerweise nur eine oder zwei eindeutige Blockkennungen in jedem Satz von vier Cache-Lesekennungen erscheinen. Wie oben erläutert wurde, werden benachbarte Pixel häufig auf die gleichen zwei Tabellen für die MIP-Tabelle abbilden, was erfordert, daß Lesevorgänge des Caches kontinuierlich zwischen den Cache-Blöcken, die die zwei Tabellen speichern, schalten, wenn Pixel eines Objekts aufbereitet werden und eine, wie oben dargelegt wurde, trilineare Interpolation verwendet ist. Bei dem veranschaulichenden Ausführungsbeispiel, das in **Fig.** 21 gezeigt ist, speichert das Mini-Verzeichnis fünf Blockkennungen, um sicherzustellen, daß, sogar wenn sich vier einzigartige Cache-Kennungen für einen gegenwärtig verarbeiteten Satz von Lesekennungen in dem Mini-Verzeichnis befinden, zumindest eine Kennung, auf die in dem vorherigen Satz von Lesekennungen zugegriffen wurde, in dem Mini-Verzeichnis verbleibt. Folglich wird, selbst wenn zwischen zwei Sätzen von vier einzigartigen Cache-Kennungen während einer trilinearen Interpolation geschaltet wird, zumindest eine der Cache-Lesekennungen für jeden Satz in dem Mini-Verzeichnis bleiben, so daß nicht vier Cache-Kennungen in einer seriellen Form gegenüber dem Hauptverzeichnis verglichen werden müssen.

[0230] Während der Aufbereitung von Texeln werden aufeinanderfolgende Lesevorgänge des Cache einen ersten Satz von vier Texeln in einer Tabelle und einen zweiten Satz von vier Texeln in einer weiteren lesen, wenn eine trilineare Interpolation verwendet ist. Während ein Grundelement aufbereitet wird, wird jeweils auf benachbarte Texel in jeder der zwei Tabellen bei jedem zweiten Zyklus zugegriffen, wobei sich zwei oder mehr der Texel im allgemeinen in einem einzelnen Cache-Block befinden. Wenn nur eine oder zwei einzigartige Kennungen in jedem Satz von vier Cache-Lesekennungen erscheinen, kann daher eine große Anzahl von Pixeln mit jeder Cache-Lesekennung, die das Mini-Verzeichnis **500** trifft, aufbereitet

werden. Wenn nur eine Cache-Lesekennung in jedem Satz von vier das Mini-Verzeichnis verfehlt, wird kein Verhaltensnachteil eingeführt, da diese Kennung gegenüber dem Hauptverzeichnis verglichen werden kann, während der nächste Satz von vier Lesekennungen mit dem Mini-Verzeichnis verglichen wird.

[0231] Es sollte offensichtlich sein, daß das Cache-Verzeichnis der vorliegenden Erfindung, das sowohl ein Hauptverzeichnis als auch ein kleineres Mini-Verzeichnis aufweist, mit vielen anderen Anwendungen verwendet werden kann und nicht auf die Verwendung in einem Texturabbildungs-Hardwaresystem begrenzt ist. Das Cache-Miniverzeichnis-Schema der vorliegenden Erfindung ist speziell beim Implementieren eines voll assoziativen Cache brauchbar und reduziert den Aufwand der Verzeichnis-Kennungsvergleiche, wenn mehrere Cache-Lesekennungen gleichzeitig verarbeitet werden, und wenn Cache-Lesekennungen mit Kennungen, auf die aufeinanderfolgend zugegriffen wird, die vorher verwendet wurden, korreliert sind. Für einen Cache-Speicher, der zu jeder Zeit X-Kennungen speichert, und bei dem N Cache-Lesekennungen gleichzeitig mit den Verzeichnis-Blockkennungen verglichen werden, ist es beispielsweise ausreichend, ein Mini-Verzeichnis beizubehalten, das M Kennungen aufweist, wobei M größer oder gleich N ist. Jede der M Mini-Verzeichniskennungen wird mit den N Cache-Lesekennungen in einer einzigen Leseoperation verglichen. Für jede Cache-Lesekennung, die in dem Mini-Verzeichnis nicht trifft, wird seriell auf das Hauptverzeichnis zugegriffen. Solche Lesekennungen werden in einem einzelnen Zyklus mit den Hauptverzeichniskennungen verglichen. Die Hardwareeinsparungen hinsichtlich der Komparatoren gegenüber einem System, bei dem jede der X Kennungen in dem Hauptverzeichnis in einer einzigen Leseoperation mit den N Lesekennungen verglichen wird, ist abhängig von dem Verhältnis von $(X+M\cdot N)/X\cdot N)$.

[0232] Der Verhaltensnachteil, der erlitten wird, um diese Hardwareeinsparungen zu erreichen, ist anwendungsabhängig, basierend auf dem Verhalten der Sequenz von Kennungen, auf die in aufeinanderfolgenden Leseoperationen zugegriffen wird. Wenn nicht mehr als eine Kennung bei jedem Lesesatz das Mini-Verzeichnis verfehlt, wird kein Nachteil eingeführt, da die den Fehlzugriff bewirkende Kennung parallel mit dem nächsten Satz von Lesekennungen, die mit dem Mini-Verzeichnis verglichen werden, mit dem Hauptverzeichnis verglichen werden kann.

[0233] Bezüglich der oben beschriebenen Komparatoren **526** bis **530**, die verwendet sind, um Verhaltensnachteile zu reduzieren, wenn zwei oder mehr Cache-Lesekennungen in dem Mini-Verzeichnis verfehlen, sind sechs verwendet, da gleichzeitig auf vier Lesekennungen zugegriffen wird. Die Anzahl der Komparatoren, die verwendet sind, um jede Cache-Lesekennung mit den anderen zu vergleichen, ist abhängig von der Anzahl N der Lesekennungen, auf die gleichzeitig zugegriffen wird, und ist gleich einer Summation von ganzen Zahlen von 1 bis N – 1.

[0234] Eine veranschaulichende Implementierung eines Cache-Verzeichnisses, das das Mini-Verzeichnis und das Hauptverzeichnis der **Fig.** 21 bis 23 aufweist, ist in **Fig.** 24 gezeigt. Es sollte offensichtlich sein, daß die Implementierung, die in **Fig.** 24 gezeigt ist, nur zu darstellenden Zwecken geliefert ist, und daß ferner andere Implementierungen verwendet werden können.

[0235] Die Mini-Verzeichniseinträge **501** bis **505** (**Fig.** 21) sind in eine Kennungskomponente, die in Kennungsregistern **501T** bis **505T** gespeichert ist, und eine Indexkomponente, die in Indexregistern **501I** bis **505I** gespeichert ist, geteilt. Wie oben erläutert wurde, empfängt das Cache-Verzeichnis einen Satz von vier Cache-Lesekennungen, die den vier Texeln (d.h. UL, UR, LL und LR) entsprechen, die am nächsten an dem Ort in einer MIP-Tabelle sind, auf den ein Pixel, auf dem gearbeitet wird, abbildet. Jede der vier Lesekennungen wird zu sechs Kennungskomparatoren **541** bis **546** geliefert. Fünf der Komparatoren (d.h. **542** bis **546**) sind jeweils ferner mit einem der fünf Mini-Verzeichniskennungsregister **501T** bis **505T** gekoppelt. Beispielsweise ist der Komparator **542** mit dem Kennungsregister **501T** für den Mini-Verzeichnis-Eintrag eins gekoppelt, und erzeugt eine Ausgabe, die anzeigt, ob die Kennung in diesem Eintrag des Mini-Verzeichnisses mit der Kennung von irgendeiner der Cache-Lesekennungen UL, UR, LL oder LR übereinstimmt. Die Komparatoren **543** bis **546** arbeiten auf eine ähnliche Art und Weise und vergleichen jeweils die Cache-Lesekennungen UL, UR, LL und LR mit den Kennungsregistern **502T** bis **505T**, die jeweils die Kennungen für die Mini-Verzeichnis-Einträge **2** bis **5** speichern. Jeder neue Satz von vier Cache-Lesekennungen wird in einem einzelnen Zyklus mit dem Mini-Verzeichnis verglichen. Am Ende dieses Zyklus werden die vier Kennungen UL, UR, LL und LR jeweils in den Registern **550** bis **553** gespeichert. Wie in **Fig.** 24 gezeigt ist, ist jedes der Register **550** bis **553** ferner mit einer Steuerschaltung **559** gekoppelt, die die Ausgaben der Mini-Verzeichniskennungskomparatoren **542** bis **546** empfängt. An dem Ende des Zyklus, in dem ein neuer Satz von vier Lesekennungen mit den Mini-Verzeichniskennungen verglichen wird, wird jedes der Register **550** bis **553** ferner mit Daten geladen, die identifizieren, ob ihre entsprechende Kennung (d.h. UL, UR, LL, LR) mit einem der Mini-Verzeichnis-Einträge übereinstimmt, und, wenn dies der Fall ist, welcher Eintrag übereinstimmte.

[0236] Wenn nur eine einzige Cache-Lesekennung einen Fehlzugriff in dem Mini-Verzeichnis bewirkt,

wird diese Kennung mit dem Hauptverzeichnis verglichen, während ein nächster Satz von vier Texellesekennungen mit dem Mini-Verzeichnis verglichen wird, wie oben erläutert wurde. Wenn ein Fehlzugriff in dem Mini-Verzeichnis auftritt, wird das Mini-Verzeichnis aktualisiert, um die Kennung einzuschließen, die den Fehlzugriff bewirkte, so daß das Mini-Verzeichnis stets die fünf Cache-Kennungen wiederspiegelt, auf die zuletzt zugegriffen wurde. Während des Zyklus, in dem eine Cache-Lesekennung, die einen Fehlzugriff in dem Mini-Verzeichnis bewirkte, mit dem Hauptverzeichnis verglichen wird, während ein nächster Satz von vier Lesekennungen mit dem Mini-Verzeichnis verglichen wird, wurden die Mini-Verzeichnis-Kennungsregister **501T** bis **505T** noch nicht aktualisiert, um die Cache-Kennung einzuschließen, die bei dem vorherigen Zyklus den Fehlzugriff in dem Mini-Verzeichnis bewirkte. Wenn der nächste Satz von Cache-Lesekennungen mit dem Mini-Verzeichnis verglichen wird, wird daher ein sechster Komparator **541** verwendet, um die vier Lesekennungen (UL, UR, LL und LR) mit der Kennung zu vergleichen, die bei dem vorherigen Zyklus den Fehlzugriff in dem Mini-Verzeichnis bewirkte und mit dem Hauptverzeichnis verglichen wird. Wenn mehr als eine einzigartige Kennung in dem Satz von vier Cache-Lesekennungen (UL, UR, LL und LR) das Mini-Verzeichnis verfehlt, wird die Pipeline durch das Cache-Verzeichnis angehalten, da mehrere Vergleiche mit dem Hauptverzeichnis stattfinden werden. Wenn jedoch nur eine einzigartige Kennung das Mini-Verzeichnis verfehlt, fährt die Pipeline auf die folgende Art und Weise fort, so daß das Cache-Verzeichnis einen neuen Satz von vier Cache-Lesekennungen bei jedem Zyklus empfängt.

[0237] Wie oben dargelegt wurde, werden die Lesekennungen, die bei dem vorherigen Zyklus mit dem Mini-Verzeichnis verglichen wurden, in den Registern **550** bis **553** gespeichert. Die Ausgänge dieser Register sind mit einem Vier-zu-Eins-Multiplexer **555** gekoppelt, der eines dieser Register zu einer Zeit auswählt, das mit dem Hauptverzeichnis verglichen werden soll und am Ende des Zyklus in das Mini-Verzeichnis geladen werden soll, so daß das Mini-Verzeichnis mit den zuletzt empfangenen Cache-Lesekennungen aktualisiert wird. Der Ausgang des Multiplexers **555** ist ferner mit dem sechsten Komparator **541** gekoppelt, so daß die Cache-Lesekennung, die bei dem vorherigen Zyklus den Fehlzugriff in dem Mini-Verzeichnis bewirkte, mit jedem des neuen Satzes von Lesekennungen UL, UR, LL und LR verglichen wird. Zusammen mit den Komparatoren **542** bis **546** stellt der Komparator **541** sicher, daß das Mini-Verzeichnis jeden Satz von vier Cache-Lesekennungen, die durch das Cache-Verzeichnis empfangen werden, mit den fünf zuletzt empfangenen Lesekennungen vergleicht.

[0238] Wie oben dargelegt wurde, wird die Cache-Lesekennungsausgabe von dem Multiplexer **555** ferner am Ende des Zyklus, in dem dasselbe mit dem Hauptverzeichnis verglichen wird, in eines der Mini-Verzeichniskennungsregister **501T** bis **505T** geladen. Folglich wird das Mini-Verzeichnis aktualisiert, um die Cache-Kennungen aufzuweisen, auf die zuletzt zugegriffen wurde,. Die Bestimmung dessen, welcher Eintrag mit der neuen Cache-Kennung von dem Multiplexer **555** beschrieben wird, wird durch ein Ersetzungsschema, das nachfolgend erläutert wird, durchgeführt.

[0239] Der Satz von sechs Komparatoren **526** bis **532**, die oben in Verbindung mit **Fig.** 23 erläutert wurden, ist der Bequemlichkeit halber in **Fig.** 24 als ein einzelner Komparatorblock gezeigt. Die Ausgaben dieser Komparatoren, ebenso wie die Ausgaben der Komparatoren **541** bis **546** werden jeweils der Steuerschaltung **559** geliefert, die mehrere Funktionen durchführt. Wenn ein Fehlzugriff auf das Mini-Verzeichnis stattfindet, bestimmt die Steuerschaltung **559**, welcher Eintrag in das Mini-Verzeichnis durch die neue Cache-Lesekennung ersetzt werden soll. Die Steuerschaltung **559** ersetzt keinen Eintrag, der durch eine der vier neu empfangenen Cache-Lesekennungen, die mit dem Mini-Verzeichnis verglichen werden, getroffen wurde, oder die letzte Cache-Lesekennung, die mit dem Hauptverzeichnis verglichen wurde, und weist diesen Einträgen eine höchste Priorität zu, um in dem Mini-Verzeichnis beibehalten zu werden. Außerdem speichert die Steuerschaltung **559** Zustandsinformationen bezüglich dessen, welche Mini-Verzeichniseinträge durch den vorhergehenden Satz von vier Lesekennungen getroffen wurden, und weist denselben die nächsthöchste Priorität, um in dem Mini-Verzeichnis beibehalten zu werden, zu. Den übrigen Einträgen wird eine geringere Priorität zugewiesen.

[0240] Die Steuerschaltung **559** wählt einen Eintrag für die Ersetzung aus, der in der Gruppe mit der geringsten Priorität ist, die zumindest einen Eintrag aufweist. Wenn zumindest ein Eintrag in der Gruppe mit der geringeren Priorität existiert, der nicht durch eine der vier neu empfangenen Cache-Lesekennungen, die mit dem Mini-Verzeichnis verglichen werden, getroffen wurde, der nicht die letzte Cache-Lesekennung, die mit dem Hauptverzeichnis verglichen wurde, war, und der nicht in dem vorhergehenden Satz von vier Lesekennungen war, wird folglich einer der Einträge in der Gruppe niedrigerer Priorität für die Ersetzung ausgewählt. Wenn jedoch keine Einträge in der Gruppe niedrigerer Priorität existieren, wird eine größere Gruppe von Einträgen ausgewählt, die nur die Einträge höchster Priorität ausschließt (d.h. diejenigen, die durch eine der vier neu empfangenen Cache-Lesekennungen getroffen wurden, und die letzte Cache-Lesekennung, die mit dem Hauptverzeichnis verglichen wurde), wobei ein Eintrag aus dieser Gruppe für die Ersetzung ausgewählt wird.

[0241] Sobald die Gruppe von verfügbaren Mini-Verzeichniseinträgen niedrigster Priorität identifiziert ist, wird eine Bestimmung dessen, welcher Eintrag in der Gruppe ersetzt werden soll, gemäß einem Ersetzungsschema durchgeführt, das zyklisch durch jeden der fünf Mini-Verzeichniseinträge läuft, jedesmal wenn einer ersetzt wird. Dies kann auf eine Anzahl von Arten durchgeführt werden. Bei einem Ausführungsbeispiel der Erfindung werden die fünf Mini-Verzeichniseinträge mit eins bis fünf bezeichnet. Der Eintrag, der ersetzt werden soll, wird aus der Gruppe mit der niedrigsten Priorität ausgewählt, indem zuerst der Eintrag mit der höchsten Nummer identifiziert wird, der nicht in der Gruppe ist, und dann der Eintrag mit der nächsthöchsten Nummer, der in der Gruppe ist, für die Ersetzung ausgewählt wird. Wenn der Eintrags nicht in der Gruppe niedrigster Priorität ist, schaltet das Schema um, so daß der Eintrag eins als der Eintrag mit der nächsthöchsten Nummer behandelt wird. Durch dieses Ersetzungsschema durchläuft die Steuerschaltung **559** zyklisch die Mini-Verzeichniseinträge, jedesmal, wenn einer ersetzt werden muß, und steuert das Laden des ausgewählten Mini-Verzeichniskennungsregisters **501T** bis **505T**.

[0242] Die Steuerschaltung **559** decodiert ferner die Ausgaben der Komparatoren **541** bis **546**, um Daten für jede der vier Lesekennungen (UL, UR, LL und LR), die anzeigen, ob die Lesekennung mit einem Eintrag in dem Mini-Verzeichnis übereinstimmte, und wenn dies der Fall ist, welcher Eintrag übereinstimmte, zu erzeugen. Diese Daten werden in dem entsprechenden Register **550** bis **553** für jede der Lesekennungen UL, UR, LL und LR gespeichert. Wenn beispielsweise die Lesekennung UL mit dem Eintrag3 des Mini-Verzeichnisses übereinstimmte, würden die Daten, die durch die Steuereinheit **559** decodiert werden, in dem UL-Register **550** gespeichert werden, um anzuzeigen, daß diese Lesekennung mit dem Eintrag3 des Mini-Verzeichnisses übereinstimmte. Wie nachfolgend erläutert wird, werden diese Daten durch die Cache-Verzeichnispipeline geleitet und zeigen an, daß der Blockindex für das UL-Texel in dem Register **503I** gespeichert ist, das den Blockindex für den Eintrag3 des Mini-Verzeichnisses hält. Wenn nur eine einzigartige Kennung für den Satz von Lesekennungen UL, UR, LL und LR einen Fehlzugriff in dem Mini-Verzeichnis bewirkt, wird jedes der Register **550** bis **553**, die diese Lesekennung speichern, mit Daten geladen, die anzeigen, daß der Blockindex für die entsprechenden Texturdaten nicht in dem Mini-Verzeichnis ist. Während des nächsten Zyklus wird die Ausgabe von einem der Register **550** bis **553**, das die Kennung, die den Fehlzugriff bewirkte, speichert, mit dem Hauptverzeichnis **520** verglichen, wobei der Blockindex für die Lesekennung von dem Hauptverzeichnis in ein Register **561** geladen wird, das den Hauptverzeichnis-Blockindex speichert. Die Daten, die anzeigen, daß der Blockindex keinem Eintrag in dem Mini-Verzeichnis entspricht, werden ebenfalls in dem Register **561** gespeichert, von einer Eingabe **562**, die von dem Ausgang des Multiplexers **555** geliefert wird.

[0243] Wie oben beschrieben wurde, weist der Cache-Speicher vier Verschachtelungen A bis D auf, so daß gleichzeitig auf vier Texel zugegriffen werden kann. Der Satz von vier Texel-Lesekennungen UL, UR, LL und LR kann auf eine beliebige Art und Weise den Verschachtelungen A bis D zugeordnet sein. Die Daten, die in den Registern **550** bis **553** gespeichert sind, die identifizieren, welcher Mini-Verzeichniseintrag den Blockindex speichert, der jedem der Texel UL, UR, LL und LR zugeordnet ist, werden durch einen Tonnenschieber **563** geleitet, der gesteuert ist, um jedes der Texel UL, UR, LL und LR mit seiner entsprechenden Verschachtelung A bis D zu korreliebeschriebene Art und Weise zu adressieren.

[0244] Wenn mehr als eine des Satzes von Cache-Lesekennungen UL, UR, LL und LR eine Fehlzugriff auf das Mini-Verzeichnis bewirkt, jedoch nur eine einzige einmalige Cache-Kennung enthalten ist, wird, wie oben erläutert wurde, nur einmal auf das Hauptverzeichnis **520** zugegriffen, um den Blockindex für diese Lesekennung zu liefern. Dieser Prozeß wird ebenfalls durch die Steuerschaltung **559** gesteuert, die die Ausgaben der Komparatoren **526** bis **532** verwendet, um zu identifizieren, ob beliebige zwei der vier Lesekennungen übereinstimmen. Wenn zwei oder mehr des Satzes von vier Lesekennungen einen Fehlzugriff auf das Mini-Verzeichnis bewirken, jedoch die gleiche Cache-Kennung aufweisen, wird jedes der entsprechenden Register **550** bis **553** durch die Steuerschaltung **559** eingestellt, um anzuzeigen, daß der Blockindex nicht in irgendeinem Mini-Verzeichniseintrag enthalten ist. Wenn die Daten, die diesen Lesekennungen entsprechen, in die Verschachtelungsindex-Register **565** bis **568** geleitet werden, wird folglich jede das Hauptverzeichnis-Blockindexregister **561** auswählen, damit dasselbe durch ihren zugeordneten Verschachtelungsindex-Multiplexer **571** geleitet wird.

[0245] Die Steuerschaltung **559** stellt ferner ein Verzeichnissteuerregister **573** ein, das steuert, welches der Lesekennungsregister **550** bis **553** mit dem Hauptverzeichnis verglichen werden soll. Das Register **573** steuert den Multiplexer **555**, um eines der Register **550** bis **553** auszuwählen, das zu einer Zeit mit dem Hauptverzeichnis verglichen werden soll. Wenn mehr als eine der Lesekennungen UL, UR, LL, LR einen Fehlzugriff in dem Mini-Verzeichnis bewirkt, jedoch eine gemeinsame Kennung verwendet, ist das Steuerregister **573** eingestellt, um anzuzeigen, daß nur eines der Register mit dem Hauptverzeichnis verglichen werden soll. Auf diese Weise wird nur einmal auf das Hauptverzeichnis zugegriffen, wenn der Satz von vier Cache-Lesekennungen nur eine einzige ein-

malige Kennung aufweist, die einen Fehlzugriff in dem Mini-Verzeichnis bewirkt.

[0246] Wenn der Satz von vier Cache-Lesekennungen (UL, UR, LL, LR) mehr als eine einzigartige Kennung aufweist, die einen Fehlzugriff in dem Mini-Verzeichnis bewirkt, wird der oben beschriebene Fluß durch die Cache-Verzeichnispipeline geändert, wobei das Cache-Verzeichnis belegt wird und bei dem nächsten Zyklus keinen neuen Satz von Lesekennungen empfängt. Das Verzeichnis zeigt an, daß es belegt ist, so daß jedes der Register **550** bis **553**, das eine Lesekennung aufweist, die einen Fehlzugriff in dem Mini-Verzeichnis bewirkte, mit dem Hauptverzeichnis verglichen werden kann und nicht mit einer neuen Lesekennung überschrieben werden wird. Außerdem ist der Fluß durch die Verzeichnispipeline geändert, derart, daß für jede Lesekennung, die einen Fehlzugriff in dem Mini-Verzeichnis bewirkte, auf das Hauptverzeichnis zugegriffen werden kann, und der Blockindex, der diesen entspricht, aus dem Hauptverzeichnis in eines der Register **501l** bis **505l** oder **561** geladen werden kann. Die Pipeline ist angeordnet, um zu verhindern, daß die Daten in irgendeinem der Register **550** bis **553** durch den Tonnenschieber **563** geleitet werden, bis alle Blockindizes für den Satz von Lesekennungen (UL, UR, LL, LR) entweder von dem Hauptverzeichnis gelesen wurden oder bereits in dem Mini-Verzeichnis vorliegen. Folglich ist der Satz von Texeln UL, UR, LL und LR mit den jeweiligen zugeordneten Verschachtelungen als eine Gruppe korreliert.

[0247] Wenn mehr als eine einzigartige Kennung in einem Satz von Lesekennungen einen Fehlzugriff in dem Mini-Verzeichnis bewirkt, werden die Kennungen, die einen Fehlzugriff bewirkten, seriell verarbeitet. Während des ersten Zyklus (d.h., wenn der Satz von Kennungen mit dem Mini-Verzeichnis verglichen wird) bestimmt die Steuerschaltung **559**, welcher Eintrag in dem Mini-Verzeichnis durch eine erste Lesekennung, die einen Fehlzugriff bewirkte, ersetzt werden soll, wobei das entsprechende Register **550** bis **553** mit Daten, die anzeigen, daß der Blockindex desselben in diesem Mini-Verzeichniseintrag gespeichert werden wird, geladen wird. Wenn die Ausgabe des Registers **550** bis **553**, das die erste verarbeitete Fehlzugriffkennung speichert, während eines zweiten Zyklus mit dem Hauptverzeichnis **520** verglichen wird, wird das Hauptverzeichnis-Blockindexregister **561** mit den Daten aktualisiert, die anzeigen, welches Mini-Verzeichnisindexregister **501l** bis **505l** ersetzt werden soll. Während eines dritten Zyklus wird der entsprechende Blockindex von dem Register **561** in das Register **501l** bis **505l** geladen, das dem Mini-Verzeichniseintrag entspricht, der für die Ersetzung ausgewählt ist.

[0248] Jede der nacheinander verarbeiteten einzigartigen Kennungen, die einen Fehlzugriff in dem Mini-Verzeichnis bewirkten, wird auf die gleiche Art und Weise gehandhabt, bis die letzte Fehlzugriffkennung verarbeitet wird, die eine zweite Fehlzugriffkennung sein kann, wenn nur zwei einzigartige Kennungen einen Fehlzugriff in dem Mini-Verzeichnis bewirkten, oder die eine dritte oder vierte Fehlzugriffkennung sein kann. Die letzte Fehlzugriffkennung, die durch das Cache-Verzeichnis verarbeitet wird, wird gehandhabt, als ob sie die einzige einzigartige Kennung in dem Satz von Lesekennungen ist, die einen Fehlzugriff in dem Mini-Verzeichnis bewirkt. Wenn die Verarbeitung der letzten Fehlzugriffkennung beginnt, deaktiviert das Verzeichnis das Signal, das anzeigt, daß es belegt ist, so daß es einen neuen Satz von Lesekennungen empfangen kann.

[0249] Für die letzte verarbeitete Fehlzugriffkennung lädt die Steuerschaltung **559** ihr entsprechendes Register **550** bis **553** mit Daten, die anzeigen, daß der Blockindex für die Kennung nicht in irgendeinem Mini-Verzeichniseintrag gespeichert ist. Dies kann während des ersten Zyklus durchgeführt werden, in dem alle Lesekennungen mit dem Mini-Verzeichnis verglichen werden, oder parallel zu der Verarbeitung der anderen Fehlzugriffkennungen. Während des Zyklus, in dem die letzte Fehlzugriffkennung mit dem Hauptverzeichnis verglichen wird, werden die Daten in den Registern **550** bis **553** durch den Tonnenschieber **563** geleitet und in Verschachtelungs-Steuerregister **565** bis **568** geladen, wobei der Blockindex für die Fehlzugriffkennung von dem Hauptverzeichnis in das Hauptverzeichnis-Blockindexregister **561** geladen wird. Schließlich werden in der letzten Pipelinestufe des Verzeichnisses die Ausgaben der Verschachtelungsindex-Steuerregister **565** bis **568** verwendet, um ihre entsprechenden Verschachtelungsindex-Multiplexer **571** zu steuern, so daß der Index für die letzte verarbeitete Fehlzugriffkennung von dem Hauptverzeichnis-Blockindexregister **561** geliefert wird, und der Blockindex für jede der anderen Lesekennungen in dem Satz von seinem entsprechenden Mini-Verzeichnisindexregister **501l** bis **505l** geliefert wird. Es sollte offensichtlich sein, daß durch das Zugreifen auf den Blockindex für die letzte verarbeitete Fehlzugriffkennung aus dem Hauptverzeichnis-Blockindexregister **561** ein Zyklus eingespart ist, indem nicht darauf gewartet wird, daß der Blockindex für diese Kennung in sein Mini-Verzeichnisindexregister geladen wird.

Patentansprüche

1. Verfahren zum Zugreifen auf Texturdaten (S, T) in einem Texturabbildungs-Computergrafiksystem, wobei das System einen Rostcomputer (**15**) mit einem Hauptspeicher (**17**) aufweist, der Texturdaten speichert, die eine Mehrzahl von Texeln einschließen, wobei das Verfahren folgende Schritte aufweist: Speichern zumindest eines Teils der Texturdaten in einem Lokalspeicher (**48**) des Systems;

temporäres Speichern einer begrenzten Anzahl von Texeln, auf die zuletzt aus dem Lokalspeicher (**48**) zugegriffen wurde, in einem Texeldatenpuffer (**214**);
Lesen von Texeln von vorbestimmten Orten (**214A0**-**214D1**) des Texeldatenpuffers; und
Zugreifen auf Texel aus dem Lokalspeicher (**48**), nur wenn solche Texel nicht verfügbar sind, um aus dem Texeldatenpuffer erneut gelesen zu werden.

2. Verfahren gemäß Anspruch 1, das ferner den Schritt des Bestimmens, für ein momentanes Texel, aufweist, ob aus dem Lokalspeicher (**48**) auf das Texel zugegriffen werden soll, oder ob das Texel aus dem Texeldatenpuffer (**214**) erneut gelesen werden soll.

3. Verfahren gemäß Anspruch 2, bei dem der Schritt des Bestimmens den Schritt des Vergleichens einer Adresse des momentanen Texels mit einer Adresse eines Texels, auf das vorher zugegriffen wurde, aufweist.

4. Verfahren gemäß Anspruch 2, das nach dem Schritt des Bestimmens ferner den Schritt des Schreibens eines Te xelzugriffbefehls in einen Zugriffsbefehlpuffer (**206**) aufweist, wenn das momentane Texel nicht verfügbar ist, um aus dem Texeldatenpuffer (**214**) erneut gelesen zu werden.

5. Verfahren gemäß Anspruch 2, das nach dem Schritt des Bestimmens ferner den Schritt des Schreibens eines Befehls in einen Texellesepuffer aufweist, welcher anzeigt, von welchem Ort des Texeldatenpuffers das momentane Texel gelesen werden soll.

6. Verfahren gemäß Anspruch 1, bei dem der Schritt des Lesens von Texeln von vordefinierten Orten des Texeldatenpuffers (**214**) den Schritt des Wiederlesens eines Texels von einem vorher gelesenen Ort des Texeldatenpuffers (**214**) oder des Lesens eines Texels von einem neuen Ort des Texeldatenpuffers aufweist.

7. Texturabbildungs-Computergrafiksystem mit folgenden Merkmalen:
einem Hostcomputer (**15**) mit einem Hauptspeicher (**17**), der Texturdaten speichert, die eine Mehrzahl von Texeln aufweisen;
einem Lokalspeicher (**48**), der zumindest einen Teil der Texturdaten speichert;
einer Lokalspeicher-Zugrifseinheit (**82**), die mit dem Lokalspeicher (**48**) gekoppelt ist und die auf Texel aus dem Lokalspeicher (**48**) zugreift;
einem Texeldatenpuffer (**214**), der mit der Lokalspeicher-Zugrifseinheit (**82**) gekoppelt ist, welcher eine begrenzte Anzahl von Texeln speichert, auf die zuletzt durch die Zugriffseinheit (**82**) aus dem Lokalspeicher (**48**) zugegriffen wurde; und
einem Texelinterpolator, (**76**), der mit dem Texeldatenpuffer (**214**) gekoppelt ist, welcher Texel von vorbestimmten Orten (**214A0** – **214D1**) des Texeldatenpuffers (**214**) liest;
wobei die Zugriffseinheit (**82**) auf Texel aus dem Lokalspeicher (**48**) nur zugreift, wenn solche Texel nicht verfügbar sind, um aus dem Texeldatenpuffer (**214**) erneut gelesen zu werden.

8. System gemäß Anspruch 7, das ferner eine Schaltung aufweist, die mit dem Interpolator (**76**) und der Lokalspeicher-Zugriffseinheit (**82**) gekoppelt ist, welche bestimmt, ob ein momentanes Texel verfügbar ist, um aus dem Texeldatenpuffer (**214**) erneut gelesen zu werden.

9. System gemäß Anspruch 8, bei dem die Schaltung einen Komparator (**224A**) aufweist, der eine Adresse des momentanen Texels mit einer Adresse eines Texels vergleicht, auf das zuletzt aus dem Lokalspeicher (**48**) zugegriffen wurde.

10. System gemäß Anspruch 7, das ferner einen Texelinterpolator-Befehlspuffer aufweist, der mit dem Interpolator (**76**) gekoppelt ist, welcher Befehle speichert, die anzeigen, von welchen Orten in dem Texeldatenpuffer Texel gelesen werden sollen.

11. System gemäß Anspruch 7, das ferner einen Texelzugriff-Befehlspuffer aufweist, der mit der Lokalspeicher-Zugriffseinheit (**82**) gekoppelt ist, welcher Befehle speichert, wobei jeder Befehl die Zugriffseinheit (**82**) auffordert, auf ein Texel aus dem Lokalspeicher (**48**) zuzugreifen.

12. System gemäß Anspruch 7, bei dem der Lokalspeicher (**48**) eine Mehrzahl von Verschachtelungen aufweist, bei dem die Lokalspeicher-Zugriffseinheit eine Mehrzahl von Steuerungen (**50A** – **50E**) aufweist, wobei eine Steuerung jeder Verschachtelung zugeordnet ist, und bei dem die Steuerungen parallel arbeiten, um getrennt auf Daten aus jeder Verschachtelung zuzugreifen.

13. System gemäß Anspruch 12, das ferner eine Mehrzahl von Texeldatenpuffern (**214**) aufweist, wobei zumindest ein Texeldatenpuffer jeder Verschachtelung zugeordnet ist, und bei dem jede Steuerung separat auf ein Texel aus dem Lokalspeicher (**48**) zugreift, nur wenn das Texel nicht verfügbar ist, um aus einem zugeordneten Texeldatenpuffer (**214**) erneut gelesen zu werden.

14. System gemäß Anspruch 13, bei dem jeder Verschachtelung des Lokalspeichers (**48**) zwei Texeldatenpuffer zugeordnet sind.

15. System gemäß Anspruch 14, bei dem der Lokalspeicher (**48**) vier Verschachtelungen aufweist.

Es folgen 25 Blatt Zeichnungen

Anhängende Zeichnungen

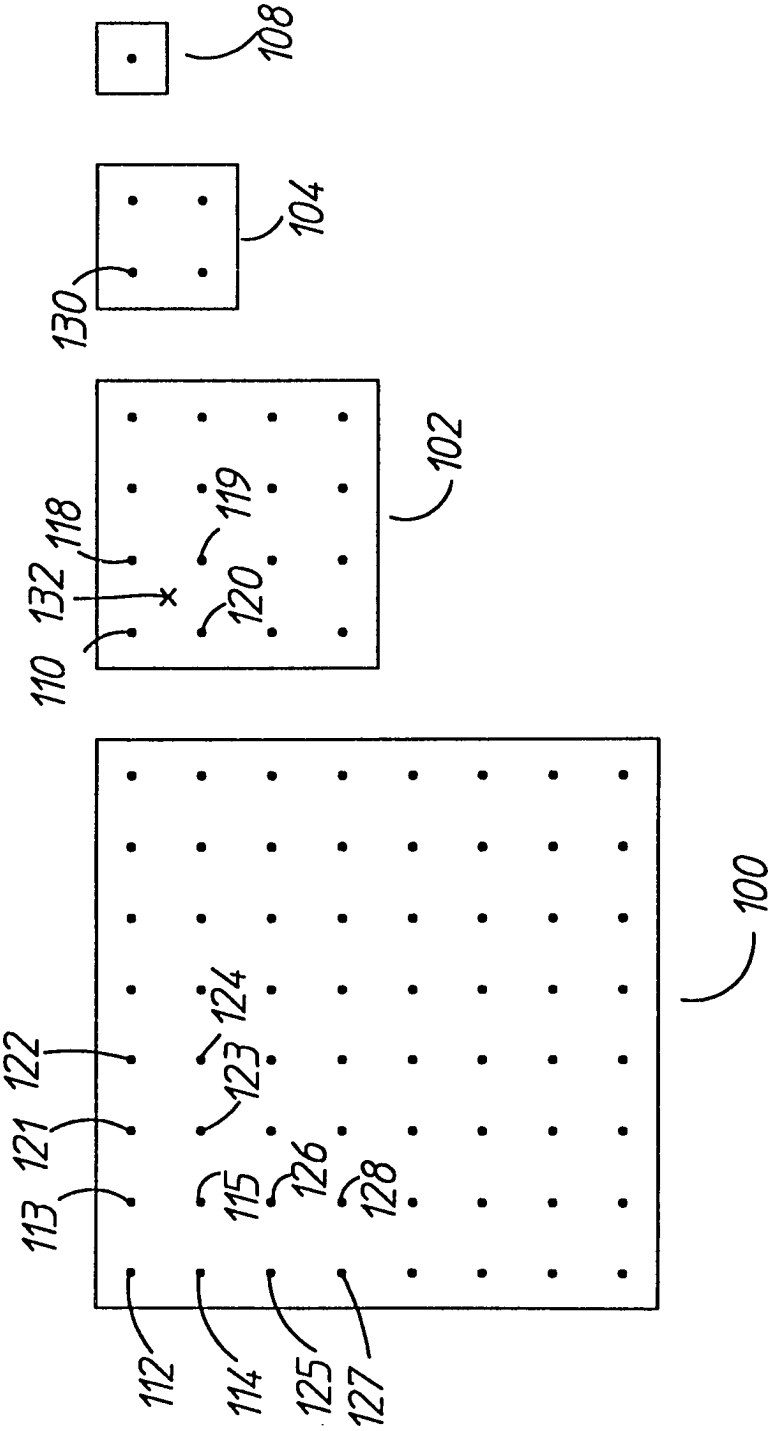


FIG 1

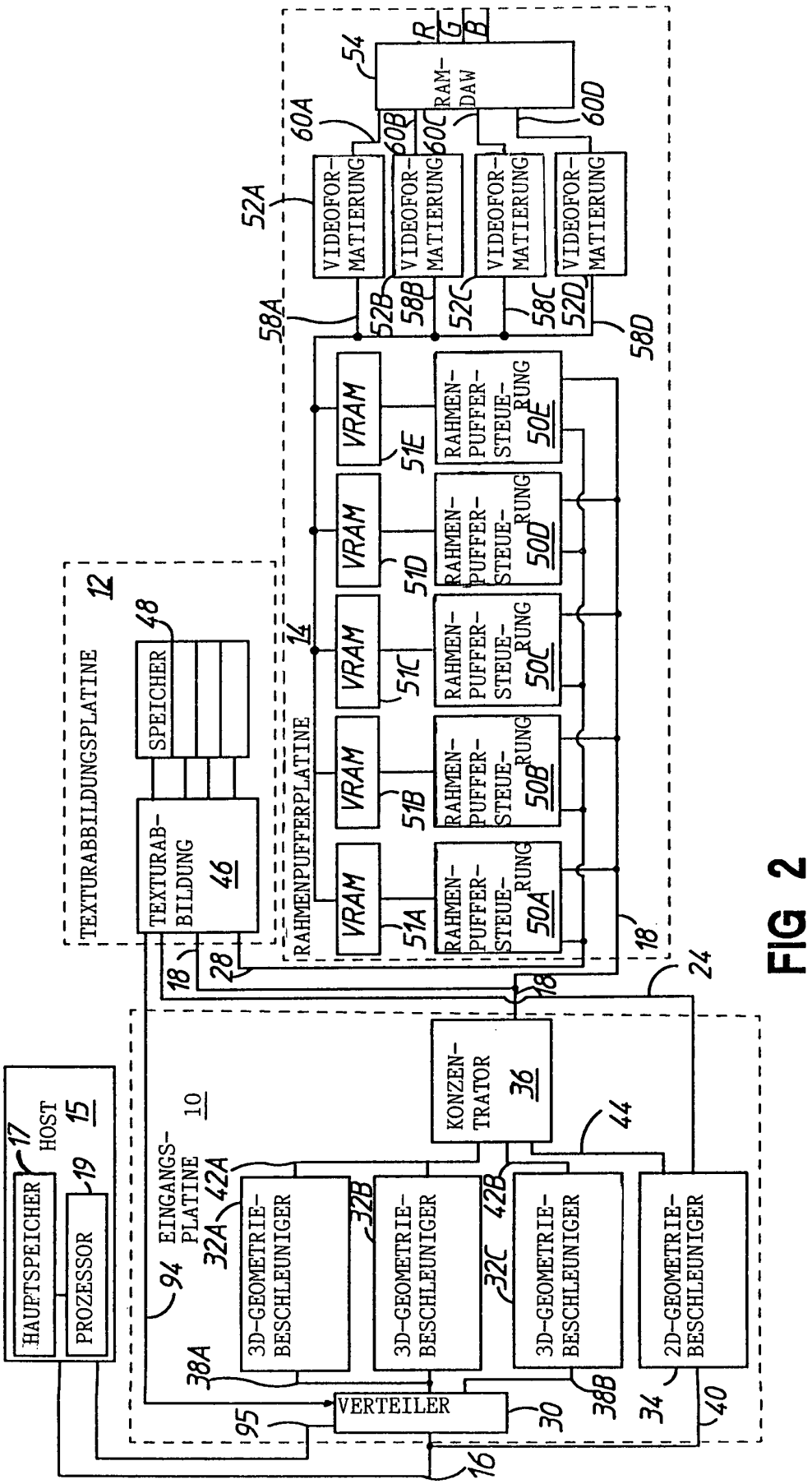
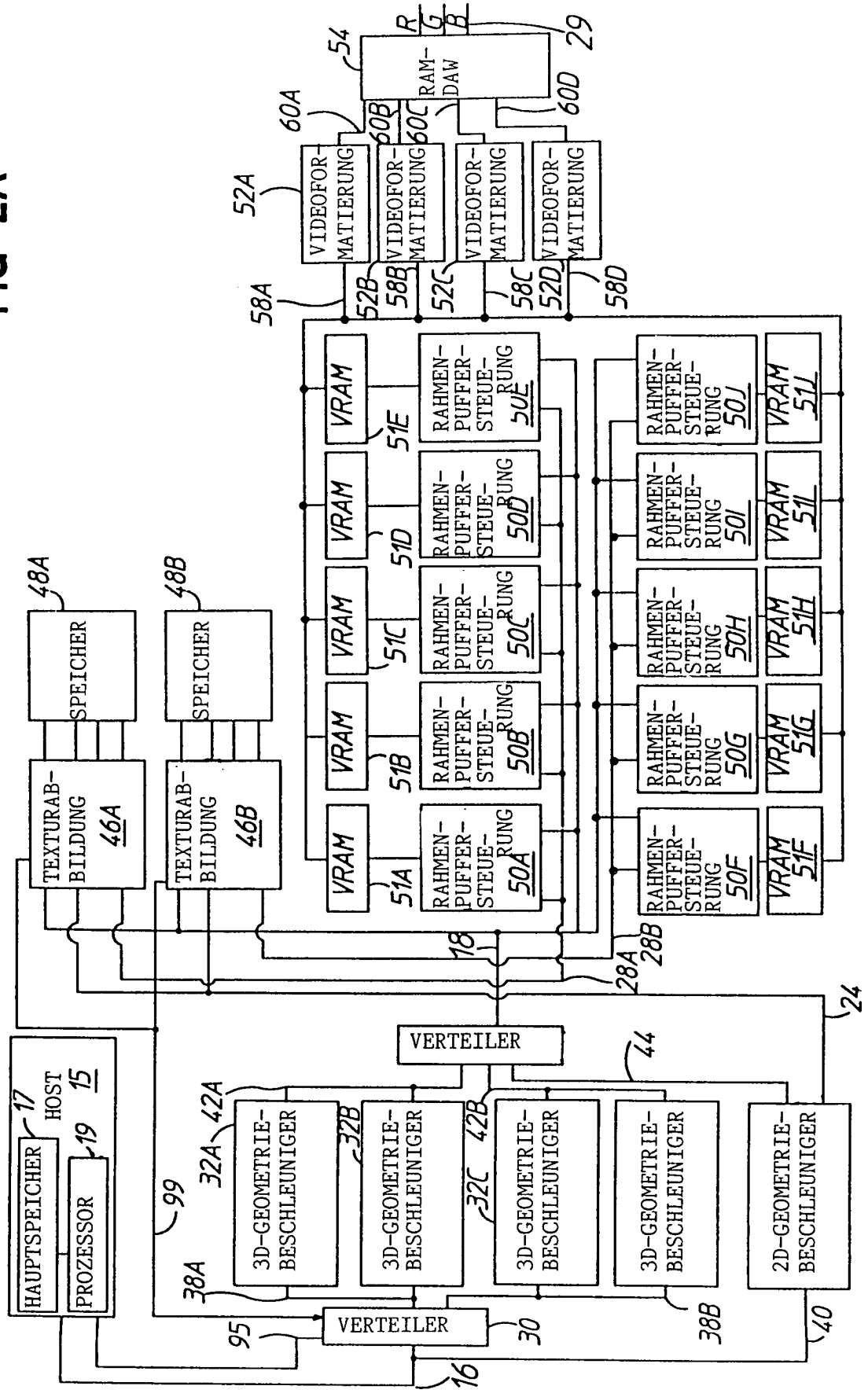

HAUPTSPEICHER
17
HOST
15
PROZESSOR
19
EINGANGS-PLATINE
10
94
3D-GEOMETRIE-BESCHLEUNIGER
32A
42A
3D-GEOMETRIE-BESCHLEUNIGER
32B
3D-GEOMETRIE-BESCHLEUNIGER
32C
42B
2D-GEOMETRIE-BESCHLEUNIGER
34
38A
38B
VERTEILER
95
30
40
16
KONZEN-TRATOR
36
44
24
18
28
TEXTURABBILDUNGSPLATINE
12
TEXTURAB-BILDUNG
46
SPEICHER
48
RAHMENPUFFERPLATINE
14
VRAM
51A
VRAM
51B
VRAM
51C
VRAM
51D
VRAM
51E
RAHMEN-PUFFER-STEUE-RUNG
50A
RAHMEN-PUFFER-STEUE-RUNG
50B
RAHMEN-PUFFER-STEUE-RUNG
50C
RAHMEN-PUFFER-STEUE-RUNG
50D
RAHMEN-PUFFER-STEUE-RUNG
50E
52A
52B
52C
52D
58A
58B
58C
58D
VIDEOFOR-MATIERUNG
60A
VIDEOFOR-MATIERUNG
60B
VIDEOFOR-MATIERUNG
60C
VIDEOFOR-MATIERUNG
60D
54
RAM-DAW
R
G
B

FIG 2

FIG 2A


HOST 15
HAUPTSPEICHER 17
PROZESSOR 19
99
SPEICHER 48A
SPEICHER 48B
TEXTURAB-BILDUNG 46A
TEXTURAB-BILDUNG 46B
VERTEILER
VERTEILER
3D-GEOMETRIE-BESCHLEUNIGER 42A
3D-GEOMETRIE-BESCHLEUNIGER 32B
3D-GEOMETRIE-BESCHLEUNIGER 42B
3D-GEOMETRIE-BESCHLEUNIGER
3D-GEOMETRIE-BESCHLEUNIGER
2D-GEOMETRIE-BESCHLEUNIGER
38A
38B
40
44
95
30
16
18
24
28A
28B
VRAM 51A
VRAM 51B
VRAM 51C
VRAM 51D
VRAM 51E
VRAM 51F
VRAM 51G
VRAM 51H
VRAM 51I
VRAM 51J
RAHMEN-PUFFER-STEUE-RUNG 50A
RAHMEN-PUFFER-STEUE-RUNG 50B
RAHMEN-PUFFER-STEUE-RUNG 50C
RAHMEN-PUFFER-STEUE-RUNG 50D
RAHMEN-PUFFER-STEUE-RUNG 50E
RAHMEN-PUFFER-STEUE-RUNG 50F
RAHMEN-PUFFER-STEUE-RUNG 50G
RAHMEN-PUFFER-STEUE-RUNG 50H
RAHMEN-PUFFER-STEUE-RUNG 50I
RAHMEN-PUFFER-STEUE-RUNG 50J
VIDEOFOR-MATIERUNG 52A
VIDEOFOR-MATIERUNG 52B
VIDEOFOR-MATIERUNG 52C
VIDEOFOR-MATIERUNG 52D
58A
58B
58C
58D
60A
60B
60C
60D
RAM-DAW 54
R
G
B
29

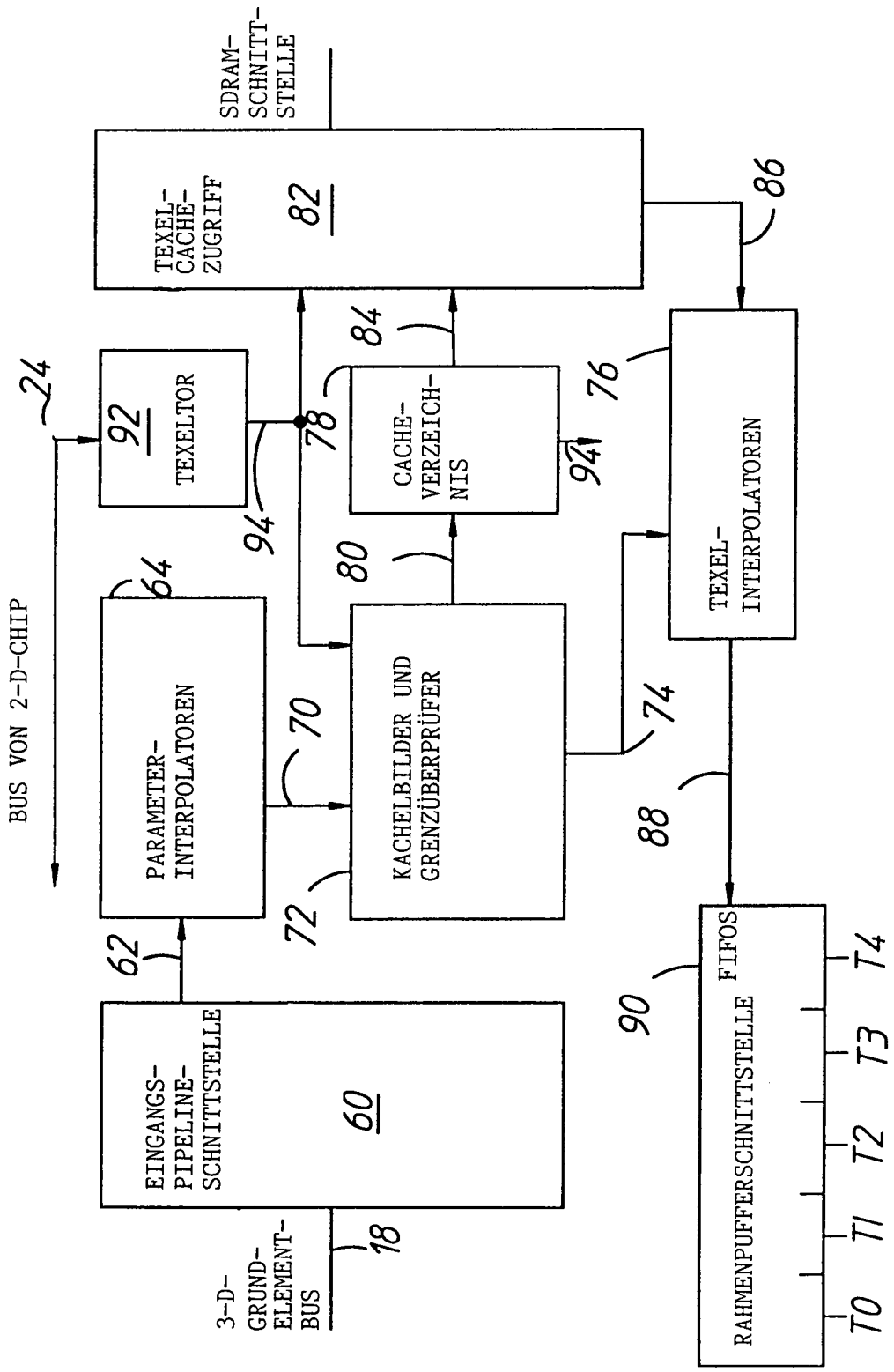

SDRAM-SCHNITT-STELLE
TEXEL-CACHE-ZUGRIFF
82
86
84
76
24
92
TEXELTOR
78
CACHE-VERZEICH-NIS
94
94
80
TEXEL-INTERPOLATOREN
BUS VON 2-D-CHIP
64
PARAMETER-INTERPOLATOREN
70
KACHELBILDER UND GRENZÜBERPRÜFER
74
72
88
62
EINGANGS-PIPELINE-SCHNITTSTELLE
60
90
FIFOS
RAHMENPUFFERSCHNITTSTELLE
T0
T1
T2
T3
T4
3-D-GRUND-ELEMENT-BUS
18

FIG 3

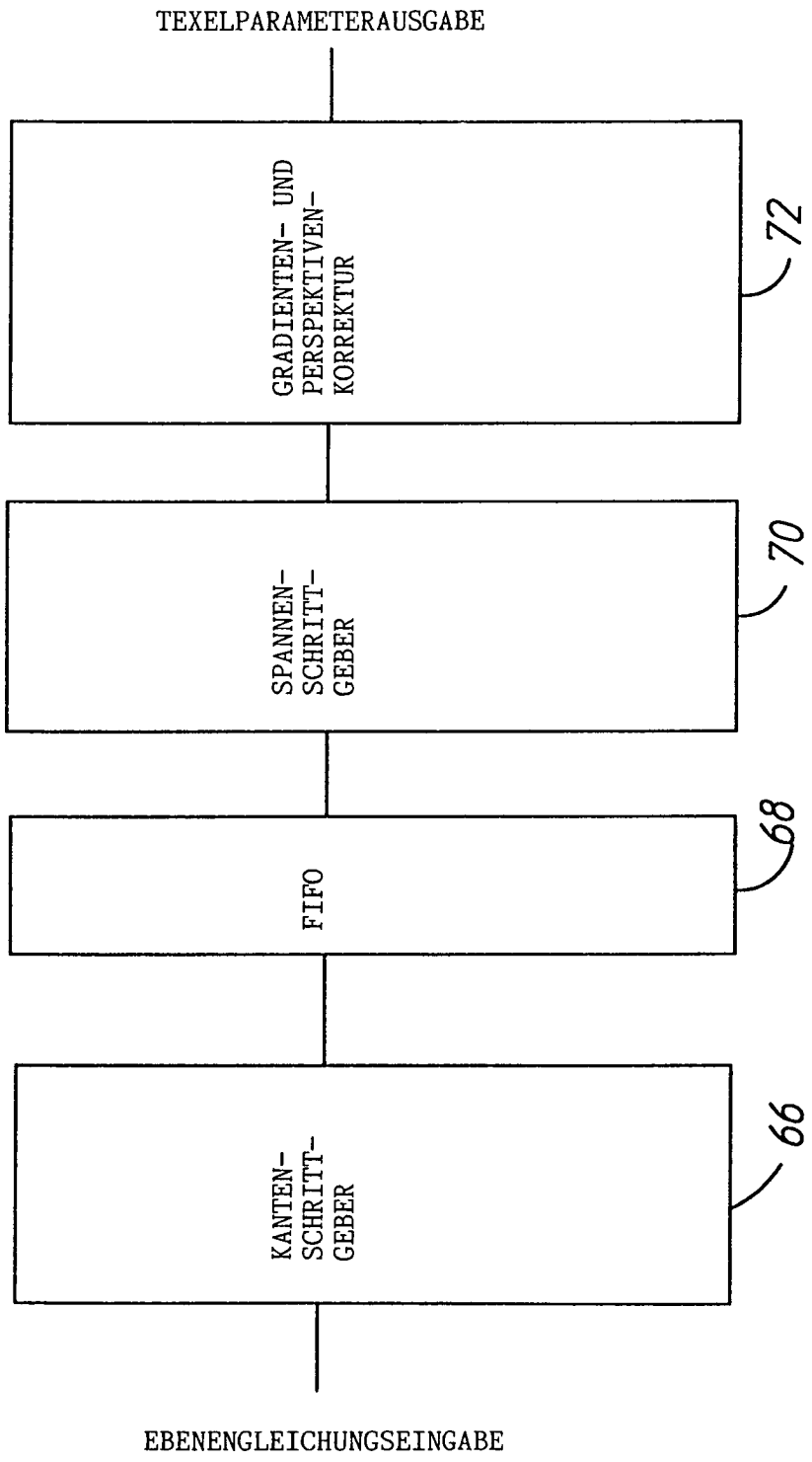

TEXELPARAMETERAUSGABE
GRADIENTEN- UND PERSPEKTIVEN-KORREKTUR
72
SPANNEN-SCHRITT-GEBER
70
FIFO
68
KANTEN-SCHRITT-GEBER
66
EBENENGLEICHUNGSEINGABE

FIG 4

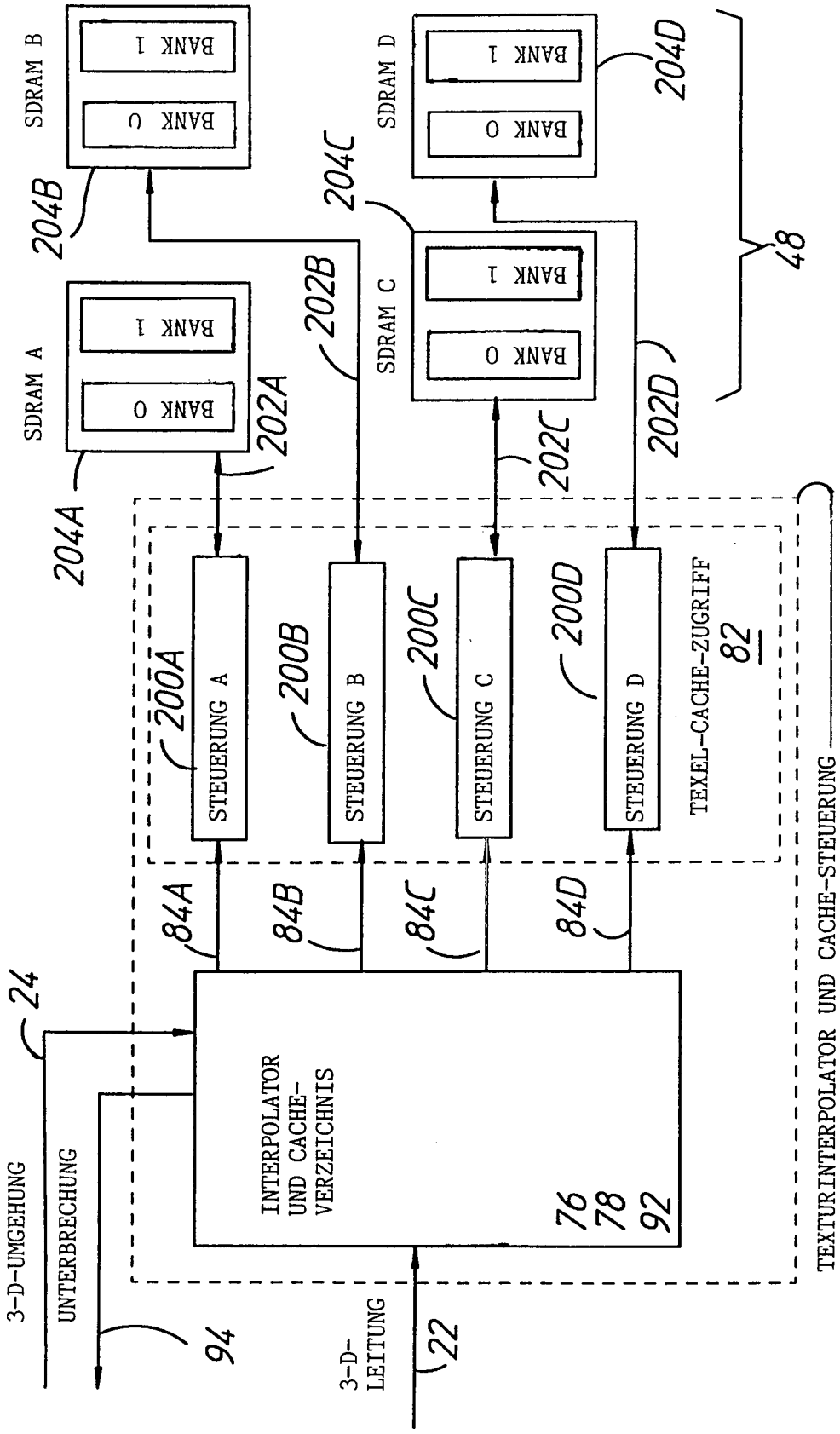
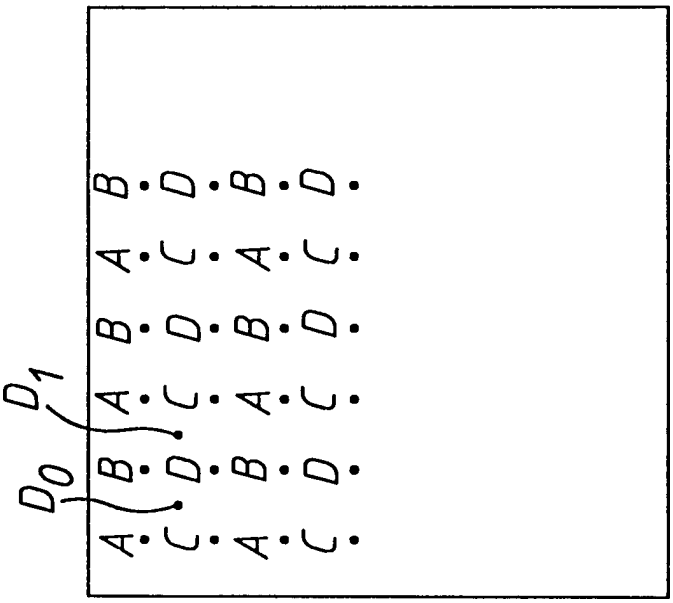

SDRAM A
BANK 0
BANK 1
204A
202A
SDRAM B
BANK 0
BANK 1
204B
202B
SDRAM C
BANK 0
BANK 1
204C
202C
SDRAM D
BANK 0
BANK 1
204D
202D
48
STEUERUNG A
200A
STEUERUNG B
200B
STEUERUNG C
200C
STEUERUNG D
200D
TEXEL-CACHE-ZUGRIFF
82
84A
84B
84C
84D
INTERPOLATOR UND CACHE-VERZEICHNIS
76
78
92
24
3-D-UMGEHUNG
UNTERBRECHUNG
94
3-D-LEITUNG
22
TEXTURINTERPOLATOR UND CACHE-STEUERUNG

FIG 5

FIG 6



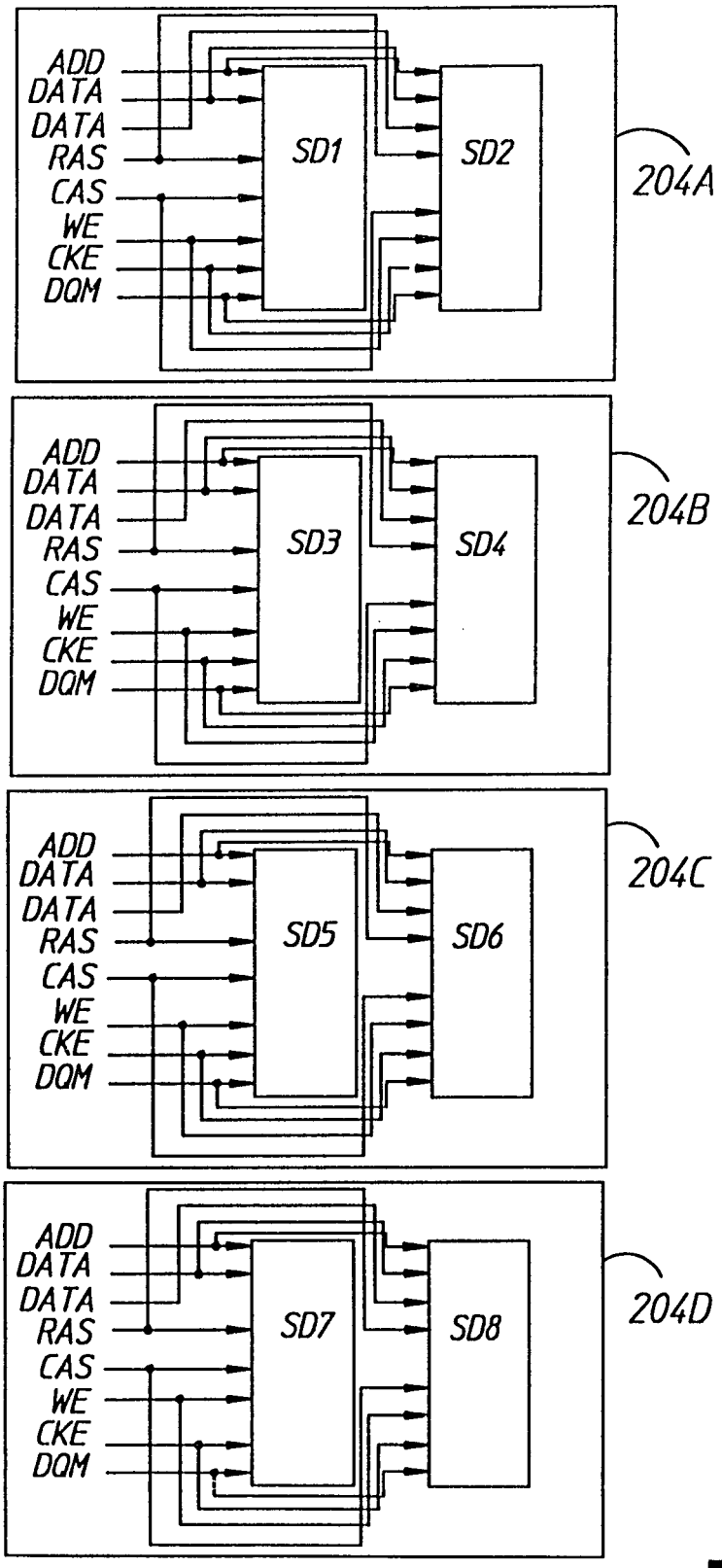

ADD
DATA
DATA
RAS
CAS
WE
CKE
DQM
SD1
SD2
204A
ADD
DATA
DATA
RAS
CAS
WE
CKE
DQM
SD3
SD4
204B
ADD
DATA
DATA
RAS
CAS
WE
CKE
DQM
SD5
SD6
204C
ADD
DATA
DATA
RAS
CAS
WE
CKE
DQM
SD7
SD8
204D

FIG 7

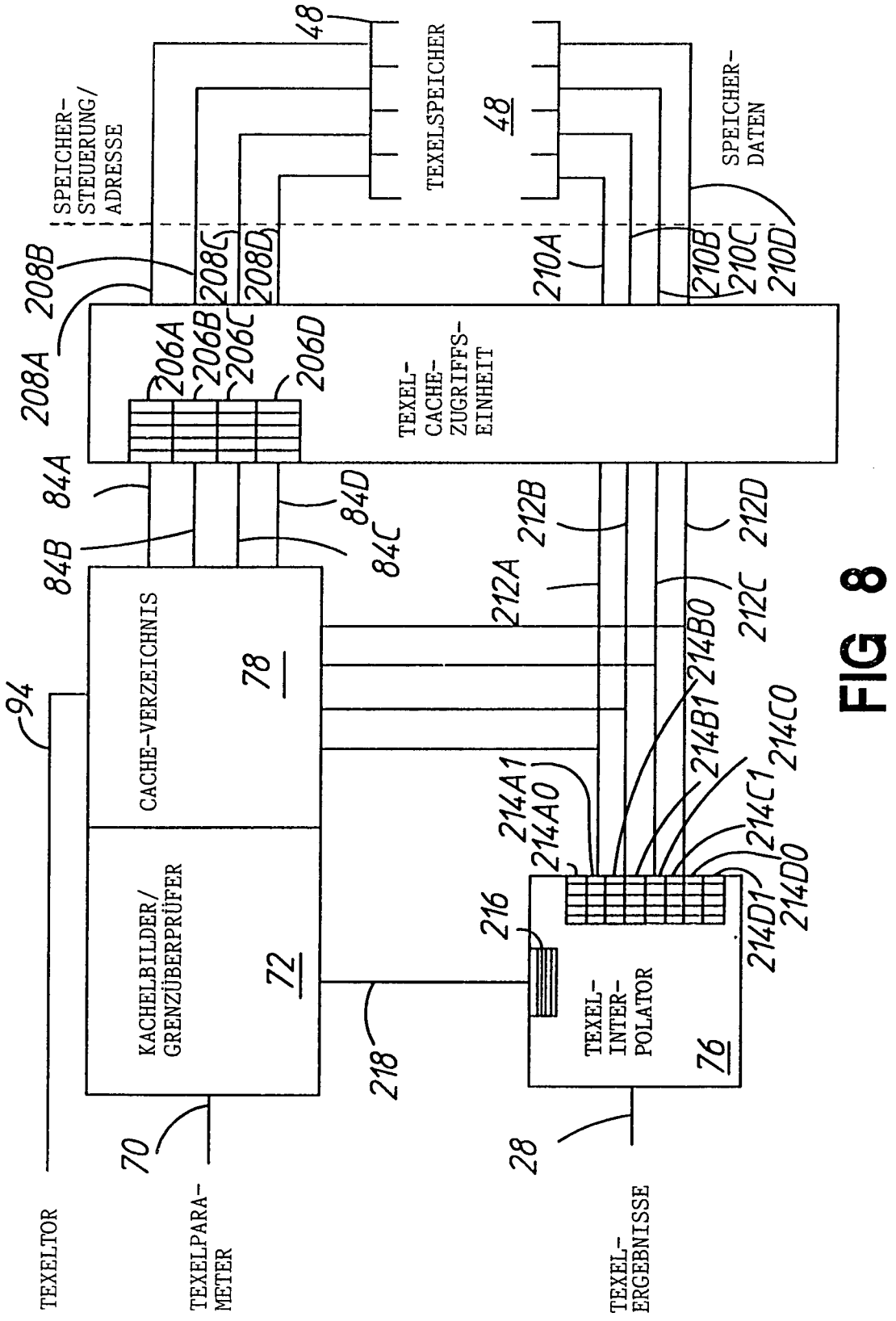

48
TEXELSPEICHER
48
SPEICHER-STEUERUNG/ ADRESSE
SPEICHER-DATEN
208B
208A
208C
208D
210A
210B
210C
210D
206A
206B
206C
206D
TEXEL-CACHE-ZUGRIFFS-EINHEIT
84A
84B
84C
84D
212A
212B
212C
212D
94
CACHE-VERZEICHNIS
78
214B0
214B1
214C0
214A1
214A0
214C1
214D0
214D1
216
KACHELBILDER/ GRENZÜBERPRÜFER
72
218
TEXEL-INTER-POLATOR
76
70
28
TEXELTOR
TEXELPARA-METER
TEXEL-ERGEBNISSE

FIG 8

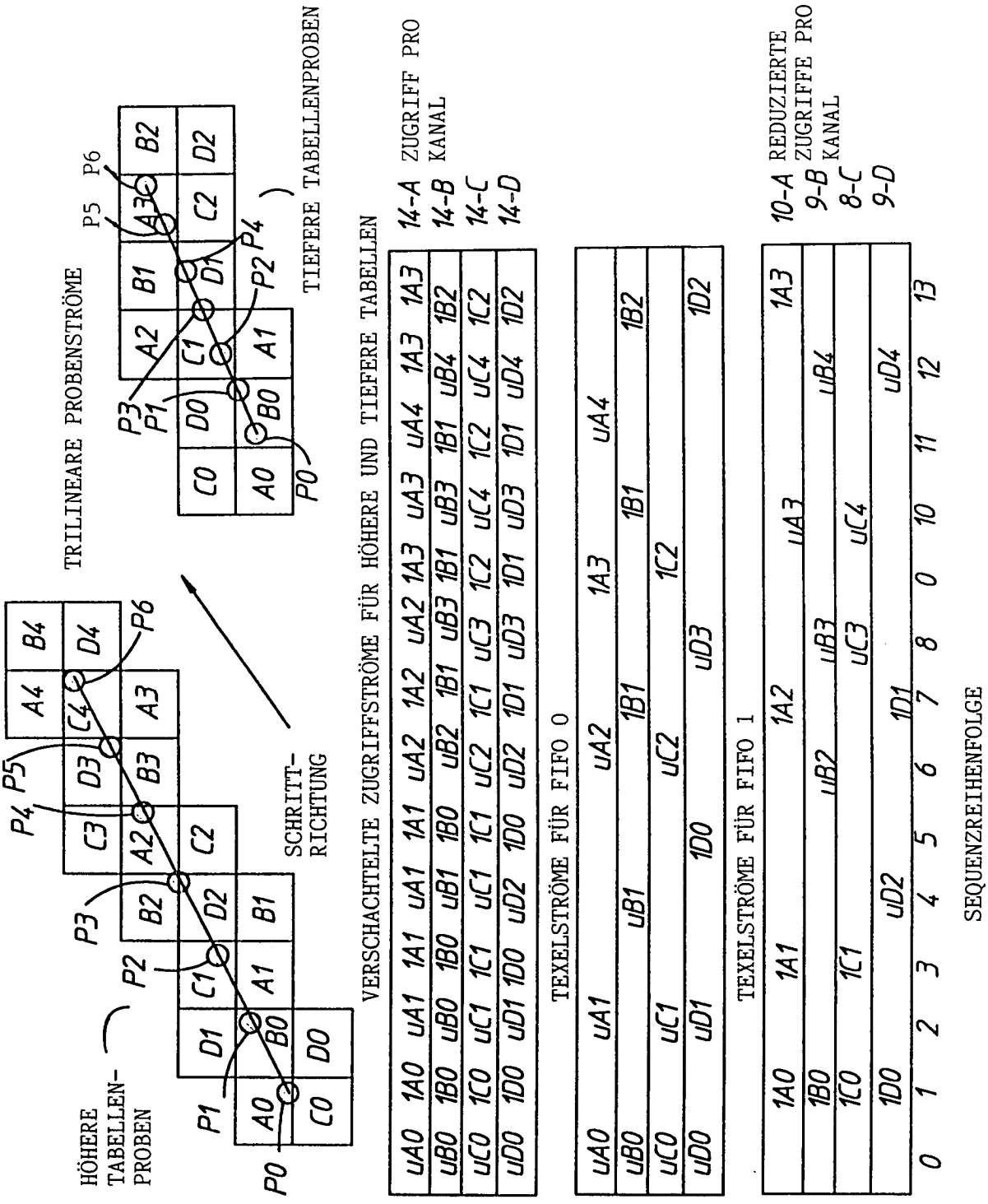

HÖHERE TABELLEN-PROBEN
TRILINEARE PROBENSTRÖME
TIEFERE TABELLENPROBEN
SCHRITT-RICHTUNG
P0
P1
P2
P3
P4
P5
P6
VERSCHACHTELTE ZUGRIFFSTRÖME FÜR HÖHERE UND TIEFERE TABELLEN
TEXELSTRÖME FÜR FIFO 0
TEXELSTRÖME FÜR FIFO 1
14-A ZUGRIFF PRO KANAL
14-B
14-C
14-D
10-A REDUZIERTE ZUGRIFFE PRO KANAL
9-B
8-C
9-D
SEQUENZREIHENFOLGE

FIG 9

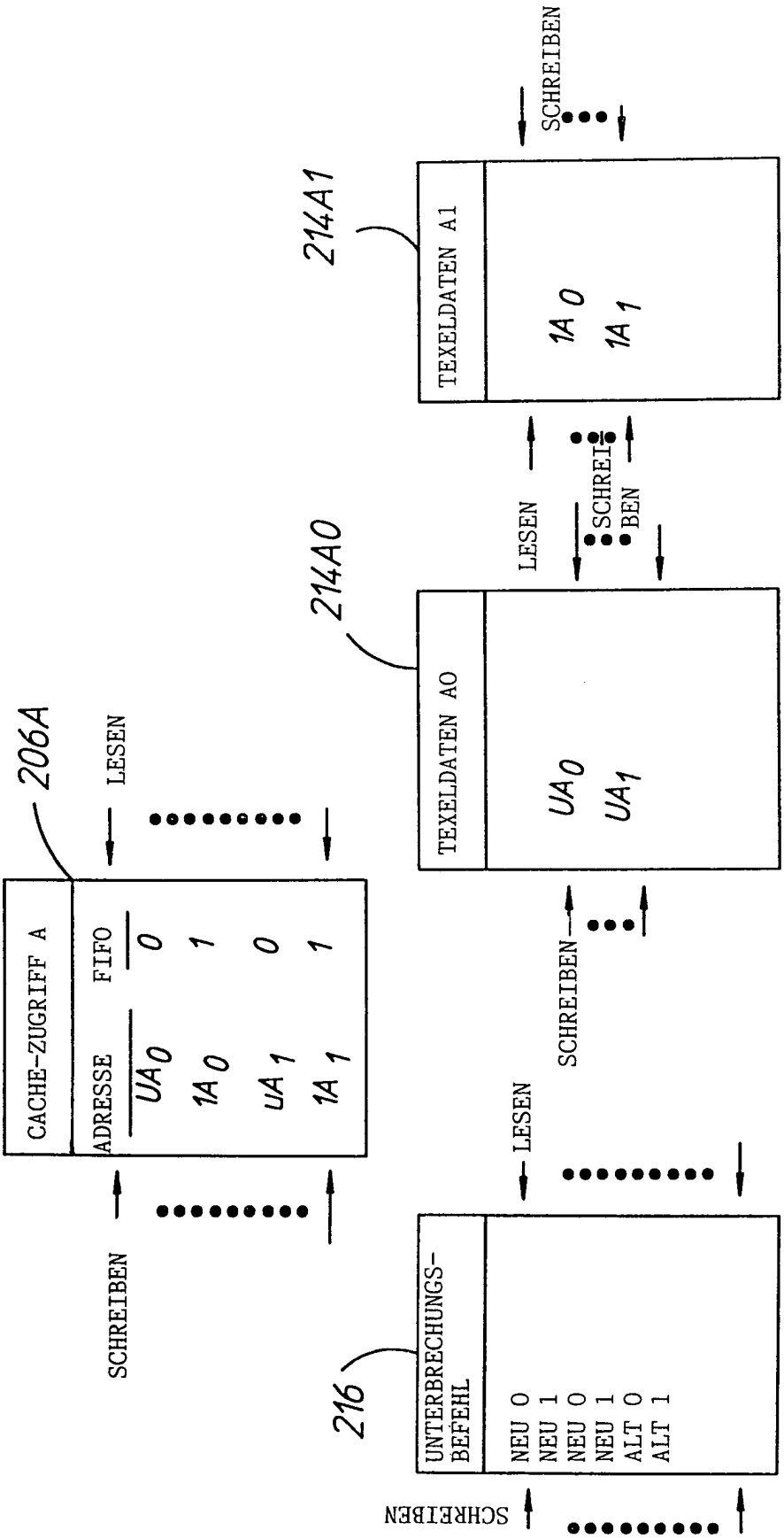

206A
CACHE-ZUGRIFF A
ADRESSE
FIFO
UA0 0
1A0 1
UA1 0
1A1 1
LESEN
SCHREIBEN
214A0
TEXELDATEN A0
UA0
UA1
214A1
TEXELDATEN A1
1A0
1A1
LESEN
SCHREIBEN
SCHREIBEN
LESEN
SCHREIBEN
216
UNTERBRECHUNGS-BEFEHL
NEU 0
NEU 1
NEU 0
NEU 1
ALT 0
ALT 1
SCHREIBEN

FIG 10

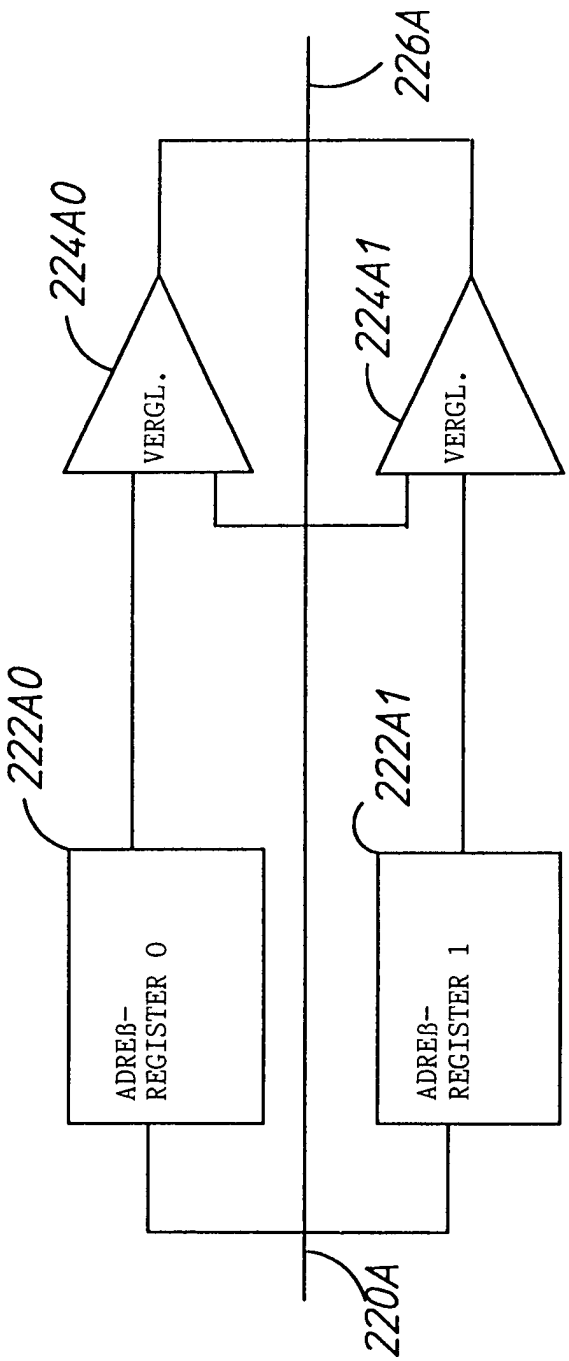

226A
224A0
224A1
VERGL.
VERGL.
222A0
222A1
ADREß-
REGISTER 0
ADREß-
REGISTER 1
220A

FIG 11

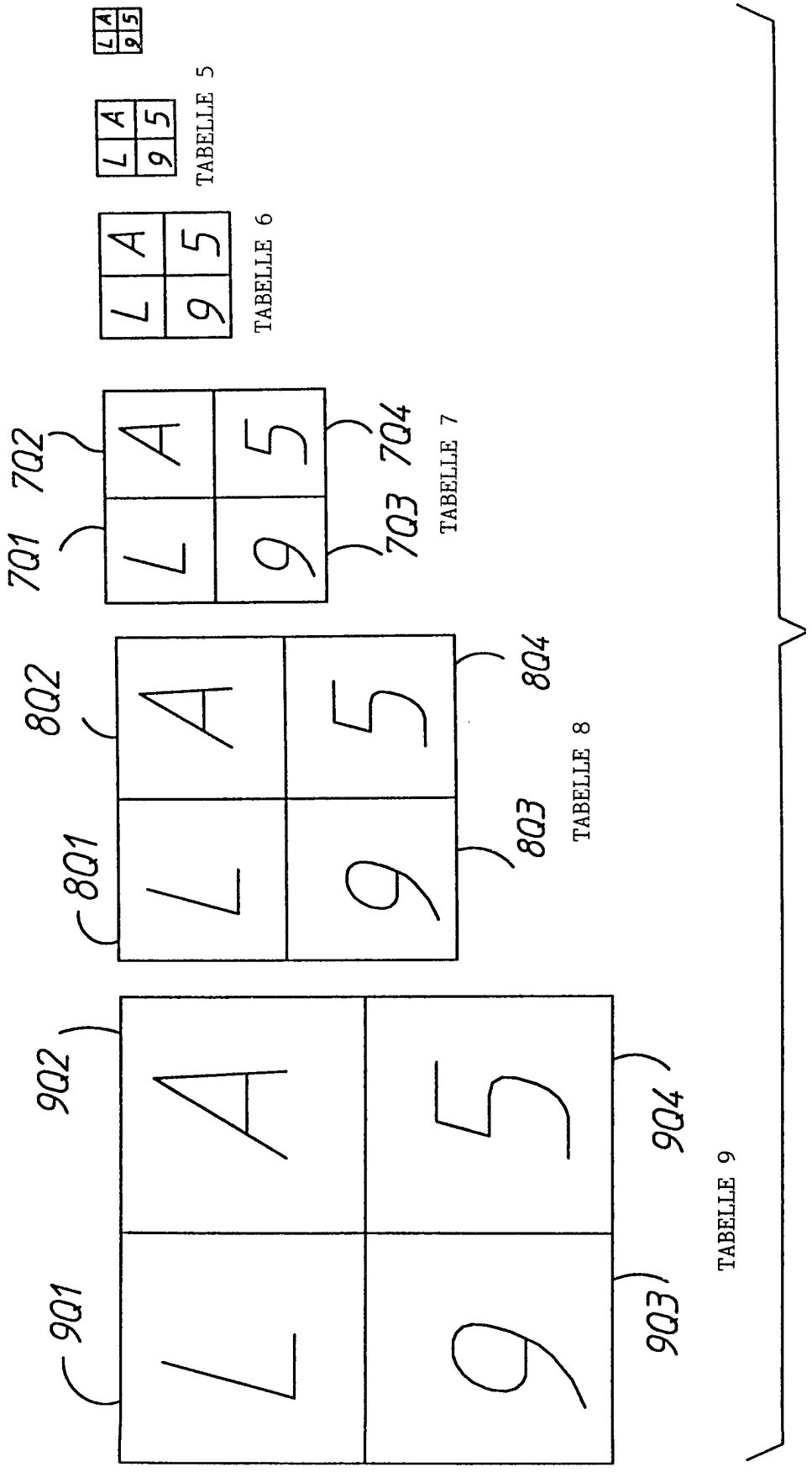

L A
9 5
TABELLE 5
L A
9 5
TABELLE 6
701
702
703
704
L A
9 5
TABELLE 7
801
802
803
804
L A
9 5
TABELLE 8
901
902
903
904
L A
9 5
TABELLE 9

FIG 12

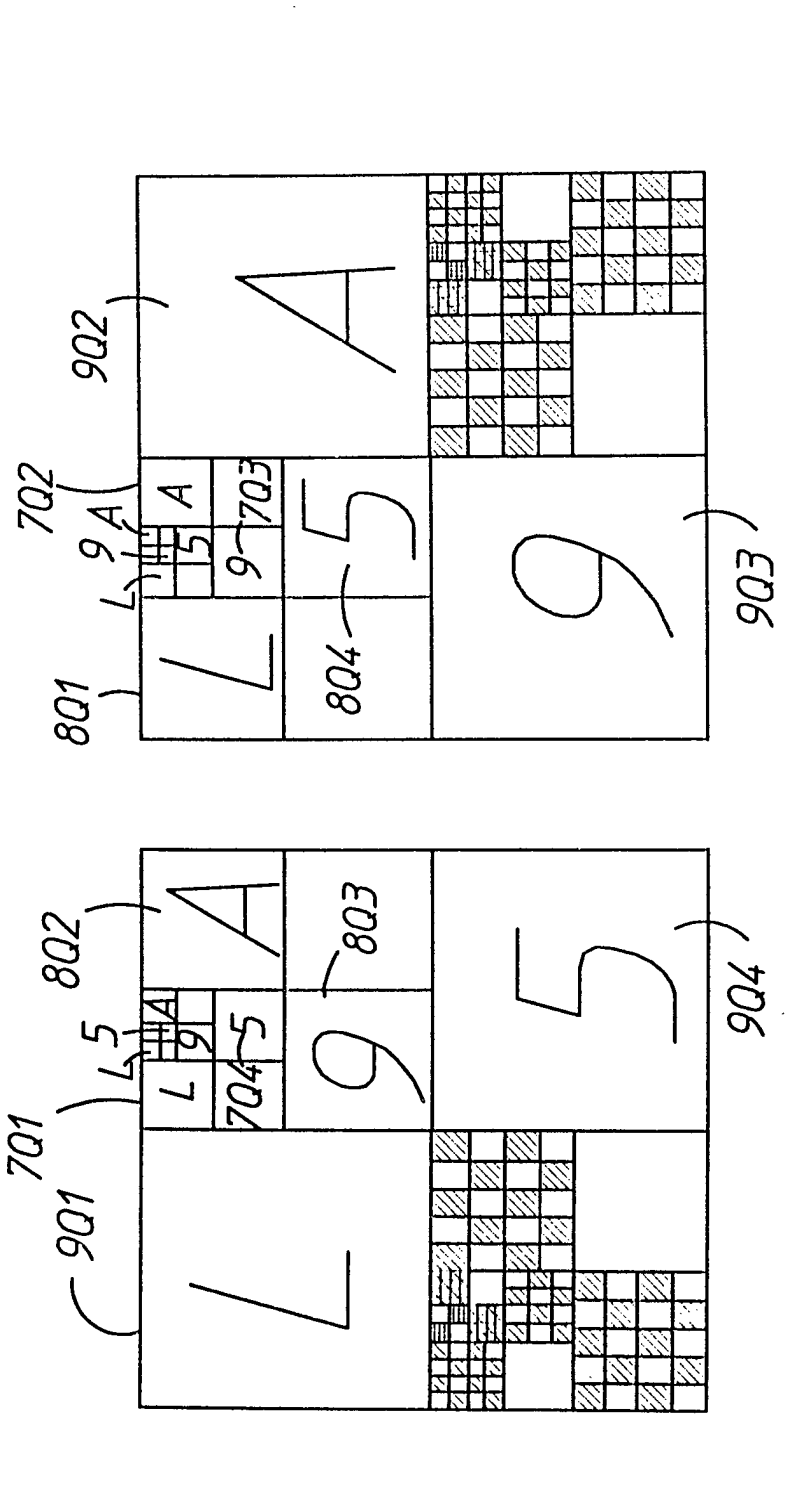


FIG 13

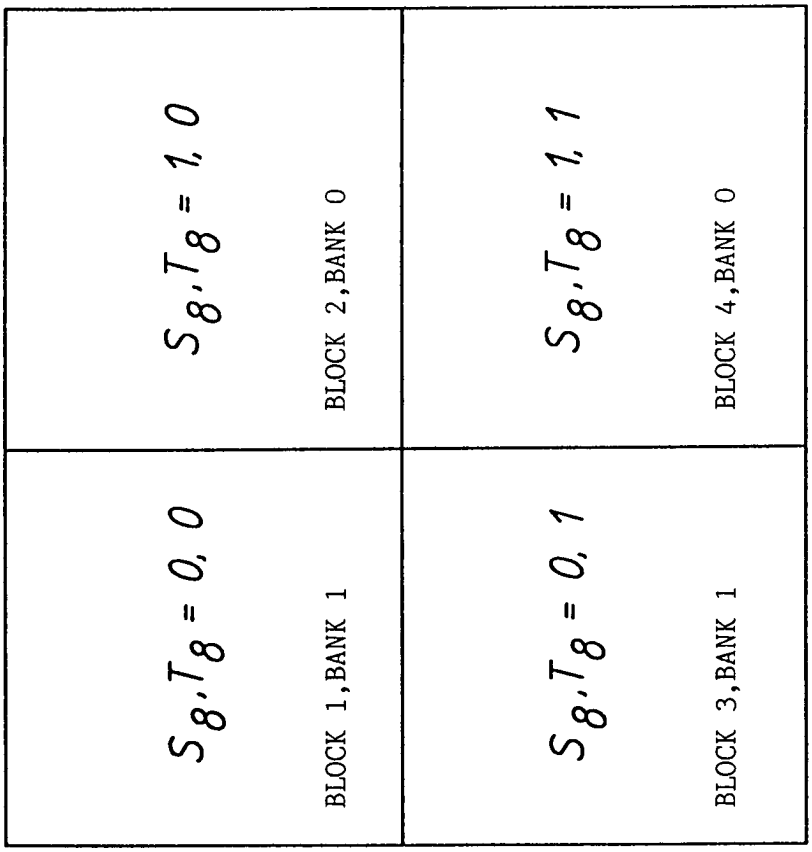
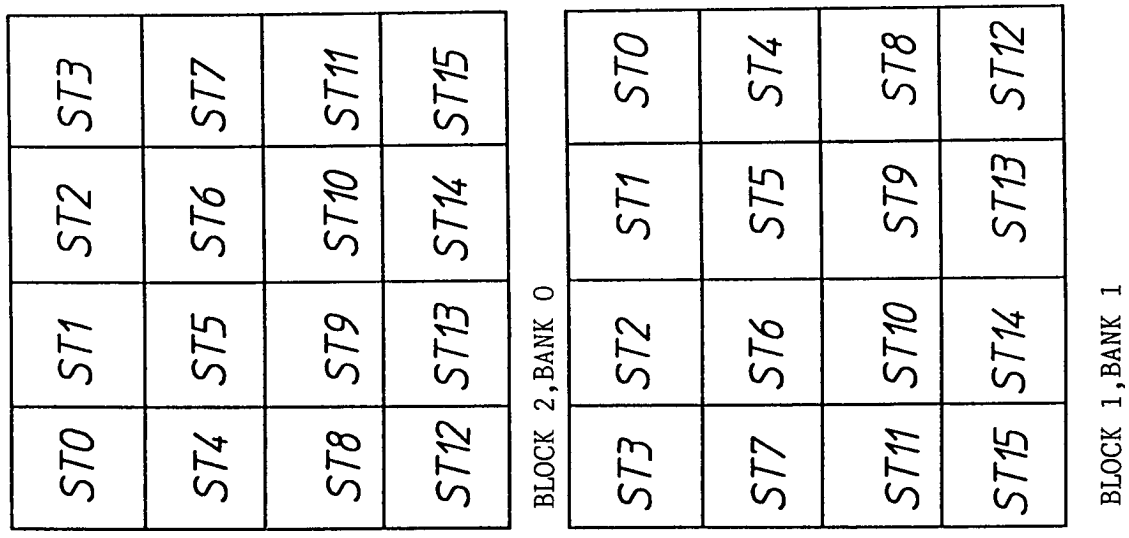
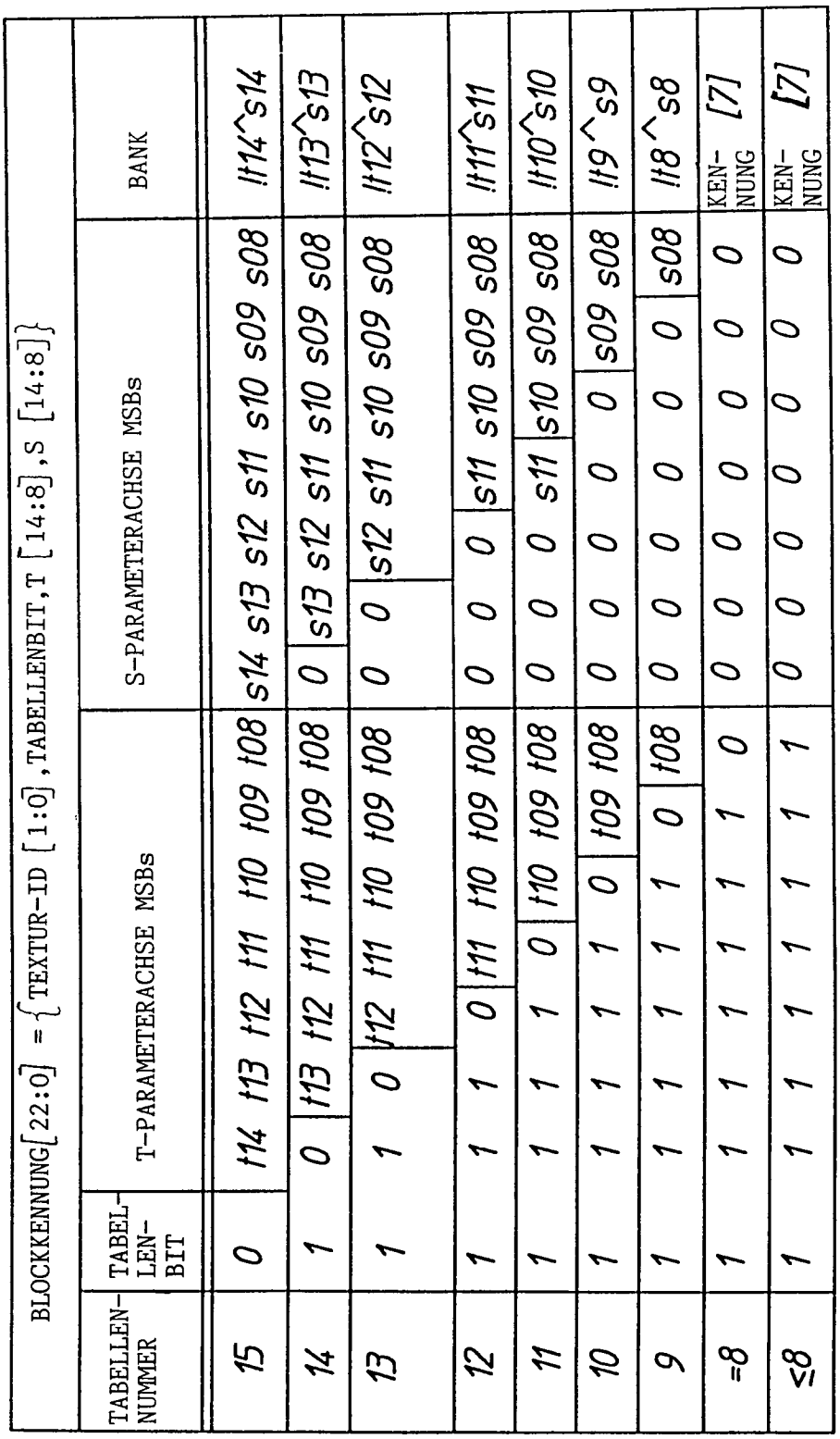


FIG 14

TABELLE 9

FIG 15


ST0 ST1 ST2 ST3
ST4 ST5 ST6 ST7
ST8 ST9 ST10 ST11
ST12 ST13 ST14 ST15
BLOCK 2, BANK 0
ST3 ST2 ST1 ST0
ST7 ST6 ST5 ST4
ST11 ST10 ST9 ST8
ST15 ST14 ST13 ST12
BLOCK 1, BANK 1



| BLOCKKENNUNG[22:0] = {TEXTUR-ID [1:0], TABELLENBIT, T [14:8], S [14:8]} | | |

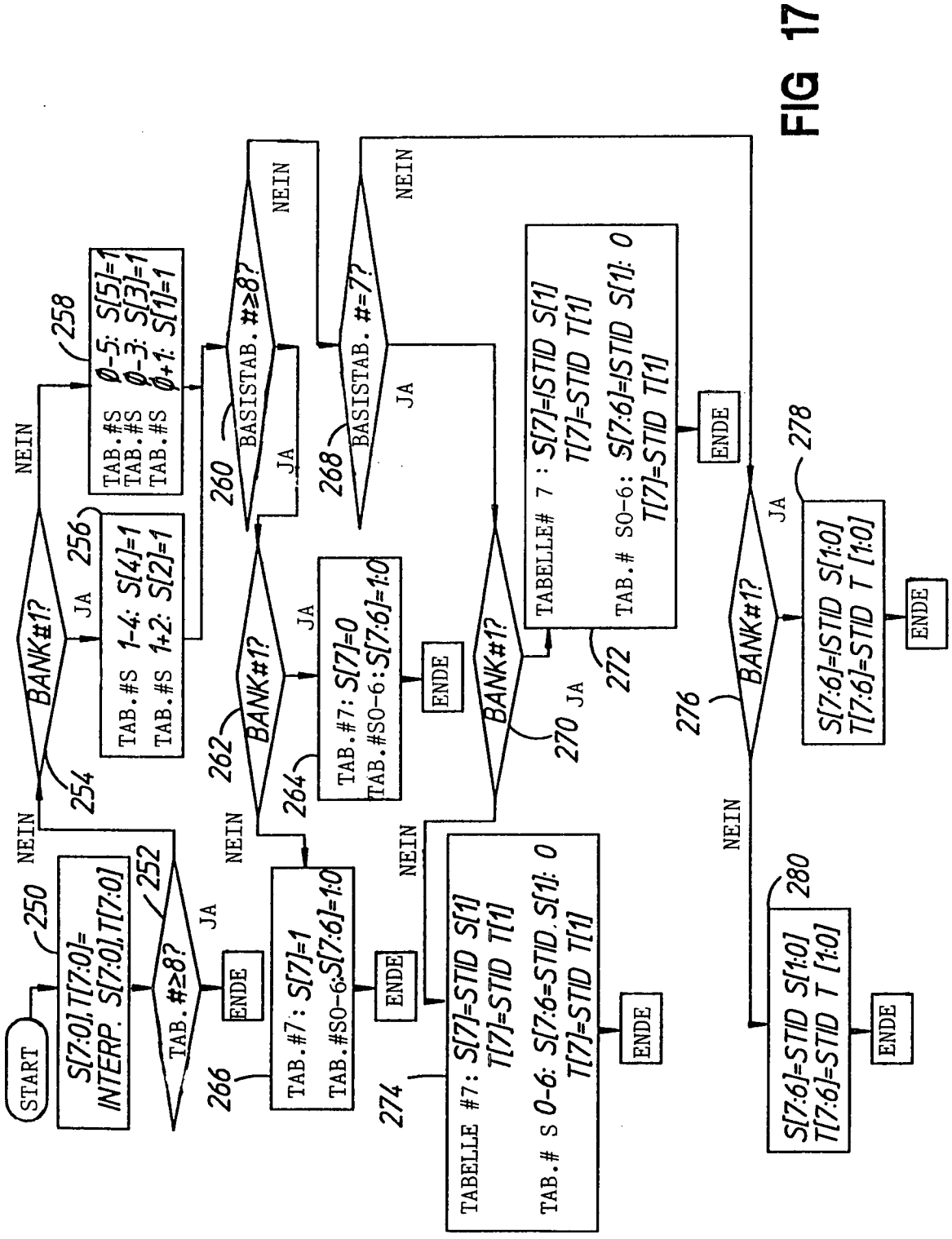

START
S[7:0].T[7:0]= INTERP. S[7:0].T[7:0]
250
TAB.#≥8?
252
JA
ENDE
NEIN
BANK#1?
254
JA
TAB.#S 1-4: S[4]=1
TAB.#S 1+2: S[2]=1
256
NEIN
TAB.#S Ø-5: S[5]=1
TAB.#S Ø-3: S[3]=1
TAB.#S Ø+1: S[1]=1
258
BASISTAB. #≥8?
260
JA
NEIN
BANK#1?
262
JA
TAB.#7: S[7]=0
TAB.#SO-6: S[7:6]=1:0
264
NEIN
TAB.#7: S[7]=1
TAB.#SO-6: S[7:6]=1:0
266
ENDE
BASISTAB. #=7?
268
JA
NEIN
BANK#1?
270
JA
TABELLE# 7 : S[7]=STID S[1] T[7]=STID T[1]
TAB.# SO-6: S[7:6]=STID S[1]: 0 T[7]=STID T[1]
272
ENDE
NEIN
TABELLE #7: S[7]=STID S[1] T[7]=STID T[1]
TAB.# S 0-6: S[7:6]=STID.S[1]: 0 T[7]=STID T[1]
274
ENDE
BANK#1?
276
JA
S[7:6]=ISTID S[1:0] T[7:6]=STID T [1:0]
278
ENDE
NEIN
S[7:6]=STID S[1:0] T[7:6]=STID T [1:0]
280
ENDE

FIG 17

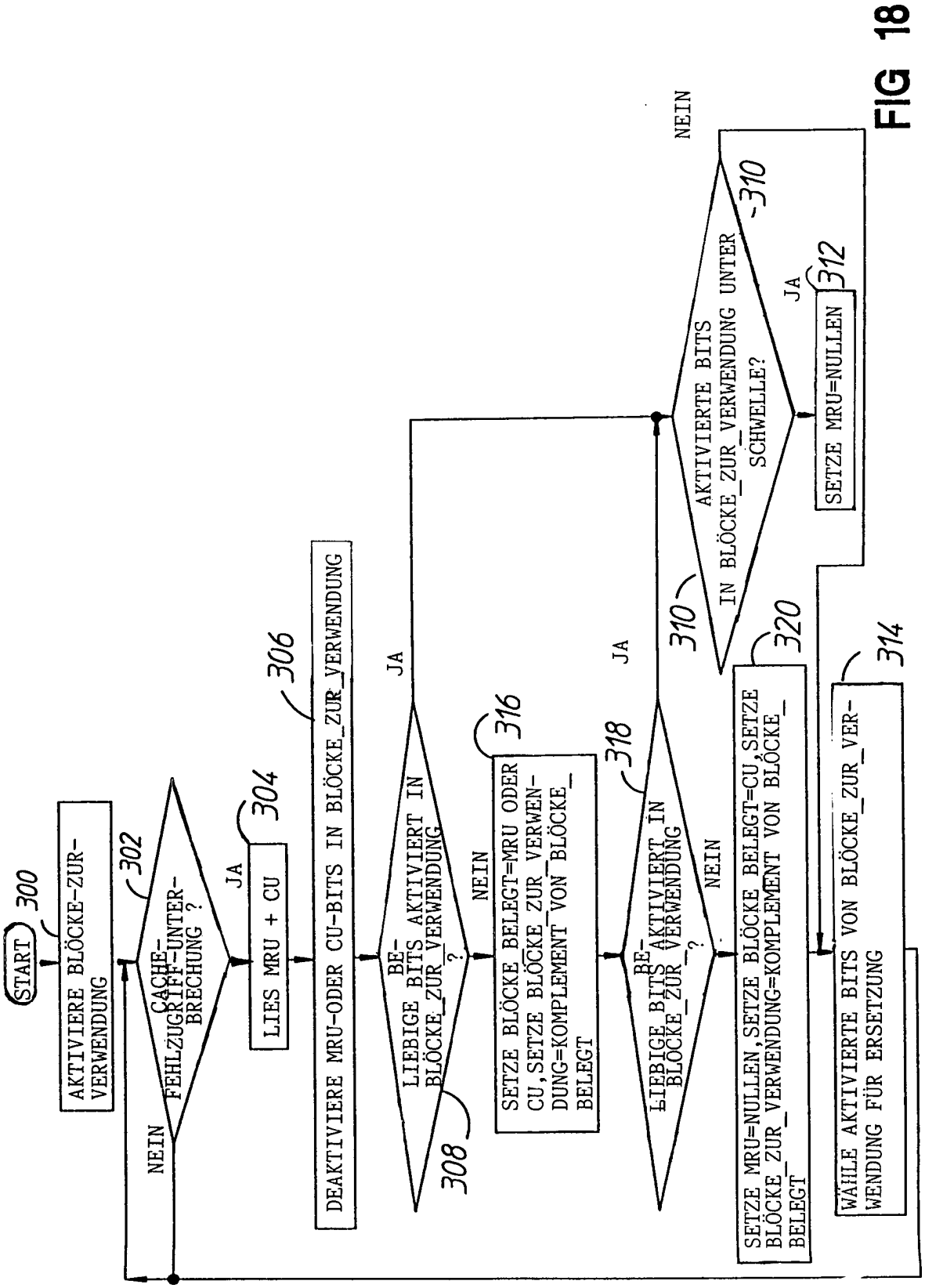

START
300
AKTIVIERE BLÖCKE-ZUR-VERWENDUNG
302
CACHE-FEHLZUGRIFF-UNTER-BRECHUNG ?
NEIN
JA
304
LIES MRU + CU
306
DEAKTIVIERE MRU-ODER CU-BITS IN BLÖCKE_ZUR_VERWENDUNG
JA
BE-LIEBIGE BITS AKTIVIERT IN BLÖCKE_ZUR_VERWENDUNG ?
308
NEIN
316
SETZE BLÖCKE BELEGT=MRU ODER CU, SETZE BLÖCKE_ZUR_VERWENDUNG=KOMPLEMENT VON BLÖCKE_BELEGT
318
JA
BE-LIEBIGE BITS AKTIVIERT IN BLÖCKE_ZUR_VERWENDUNG ?
NEIN
310
AKTIVIERTE BITS IN BLÖCKE_ZUR_VERWENDUNG UNTER SCHWELLE?
NEIN
310
JA
312
SETZE MRU=NULLEN
320
SETZE MRU=NULLEN, SETZE BLÖCKE BELEGT=CU, SETZE BLÖCKE_ZUR_VERWENDUNG=KOMPLEMENT VON BLÖCKE_BELEGT
314
WÄHLE AKTIVIERTE BITS VON BLÖCKE_ZUR_VERWENDUNG FÜR ERSETZUNG

FIG 18

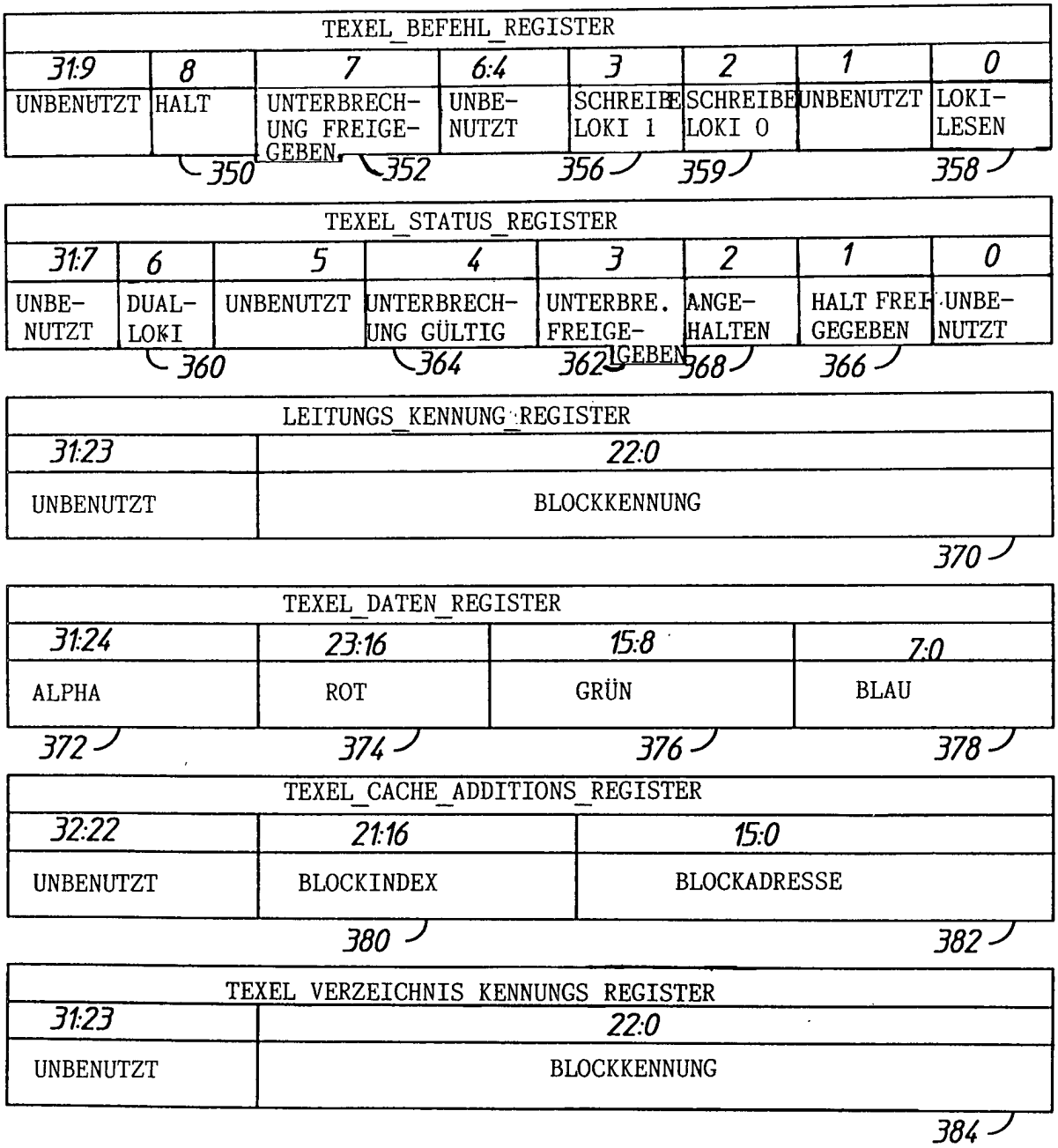

TEXEL_BEFEHL_REGISTER
31:9
UNBENUTZT
8
HALT
7
UNTERBRECH-UNG FREIGE-GEBEN
6:4
UNBE-NUTZT
3
SCHREIBE LOKI 1
2
SCHREIBE LOKI 0
1
UNBENUTZT
0
LOKI-LESEN
350
352
356
359
358
TEXEL_STATUS_REGISTER
31:7
UNBE-NUTZT
6
DUAL-LOKI
5
UNBENUTZT
4
UNTERBRECH-UNG GÜLTIG
3
UNTERBRE. FREIGE-GEBEN
2
ANGE-HALTEN
1
HALT FREI-GEGEBEN
0
UNBE-NUTZT
360
364
362
368
366
LEITUNGS_KENNUNG_REGISTER
31:23
UNBENUTZT
22:0
BLOCKKENNUNG
370
TEXEL_DATEN_REGISTER
31:24
ALPHA
23:16
ROT
15:8
GRÜN
7:0
BLAU
372
374
376
378
TEXEL_CACHE_ADDITIONS_REGISTER
32:22
UNBENUTZT
21:16
BLOCKINDEX
15:0
BLOCKADRESSE
380
382
TEXEL_VERZEICHNIS_KENNUNGS_REGISTER
31:23
UNBENUTZT
22:0
BLOCKKENNUNG
384

FIG 19

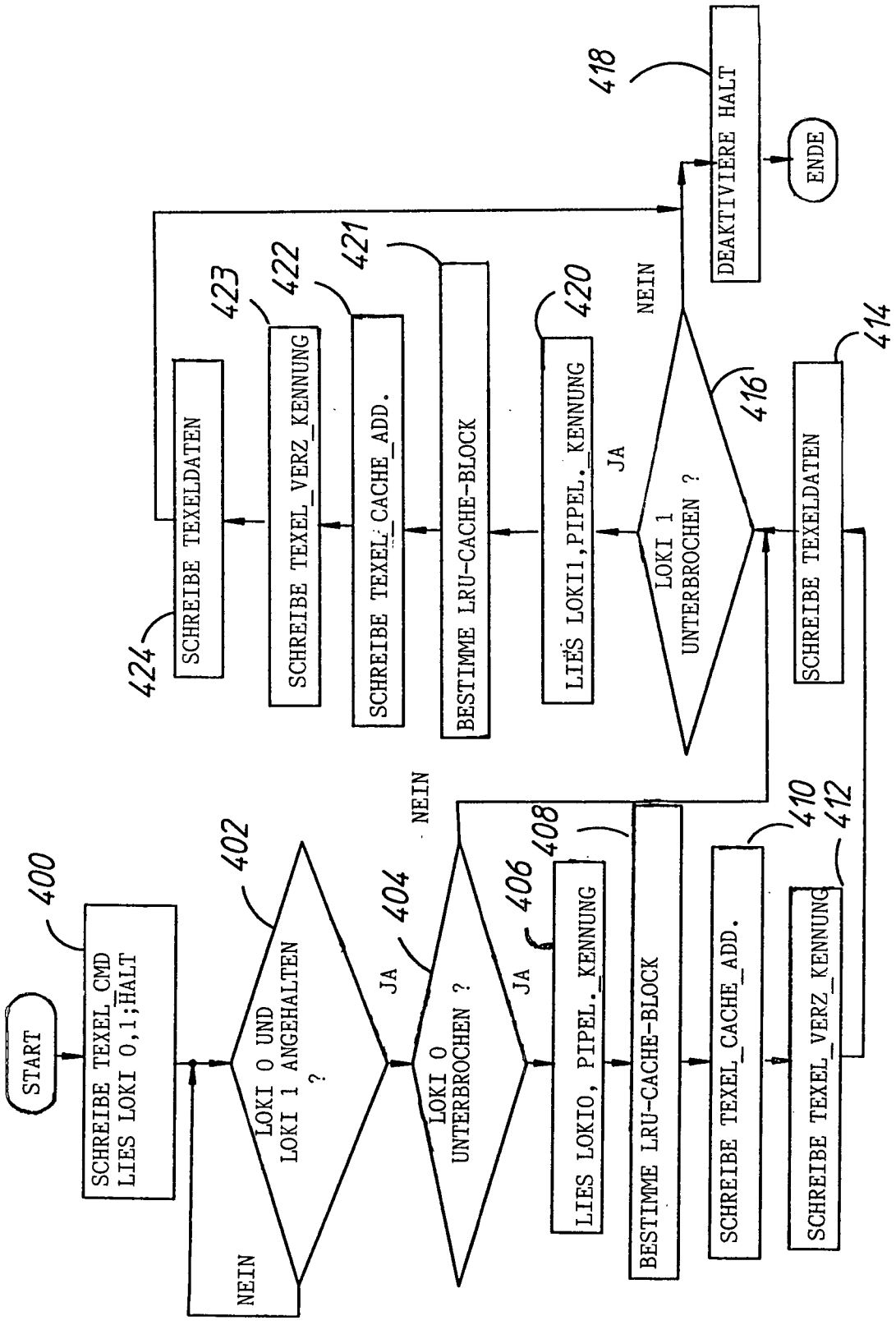

START
400
SCHREIBE TEXEL_CMD
LIES LOKI 0,1;HALT
402
LOKI 0 UND
LOKI 1 ANGEHALTEN
?
NEIN
JA
404
LOKI 0
UNTERBROCHEN ?
NEIN
JA
406
408
LIES LOKIO, PIPEL._KENNUNG
BESTIMME LRU-CACHE-BLOCK
410
SCHREIBE TEXEL_CACHE_ADD.
412
SCHREIBE TEXEL_VERZ_KENNUNG
414
SCHREIBE TEXELDATEN
416
LOKI 1
UNTERBROCHEN ?
JA
NEIN
420
LIES LOKI1,PIPEL._KENNUNG
421
BESTIMME LRU-CACHE-BLOCK
422
SCHREIBE TEXEL_CACHE_ADD.
423
SCHREIBE TEXEL_VERZ_KENNUNG
424
SCHREIBE TEXELDATEN
418
DEAKTIVIERE HALT
ENDE

FIG 20

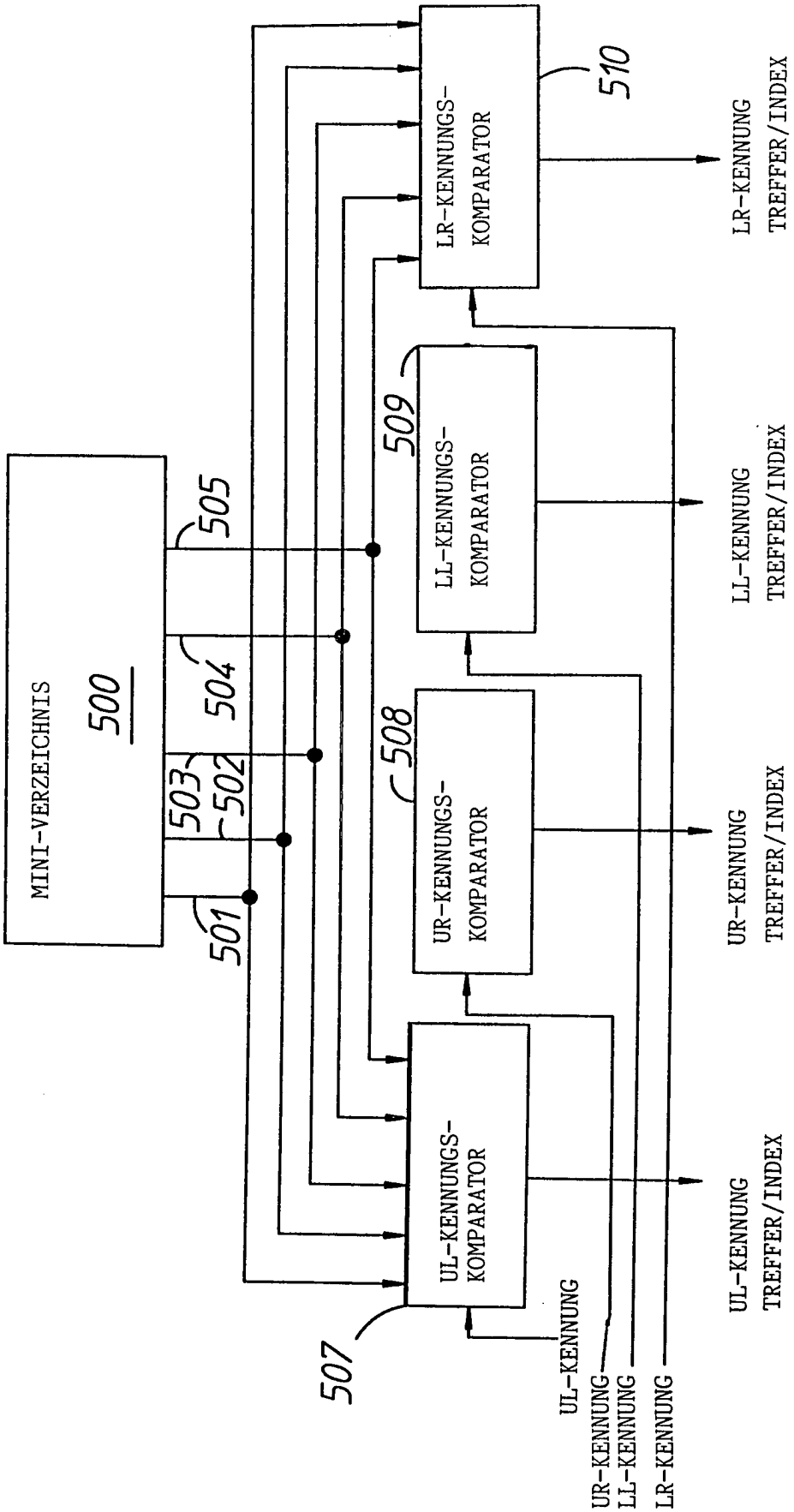

MINI-VERZEICHNIS
500
501
502
503
504
505
507
508
509
510
UL-KENNUNGS-KOMPARATOR
UR-KENNUNGS-KOMPARATOR
LL-KENNUNGS-KOMPARATOR
LR-KENNUNGS-KOMPARATOR
UL-KENNUNG
UR-KENNUNG
LL-KENNUNG
LR-KENNUNG
UL-KENNUNG TREFFER/INDEX
UR-KENNUNG TREFFER/INDEX
LL-KENNUNG TREFFER/INDEX
LR-KENNUNG TREFFER/INDEX

FIG 21

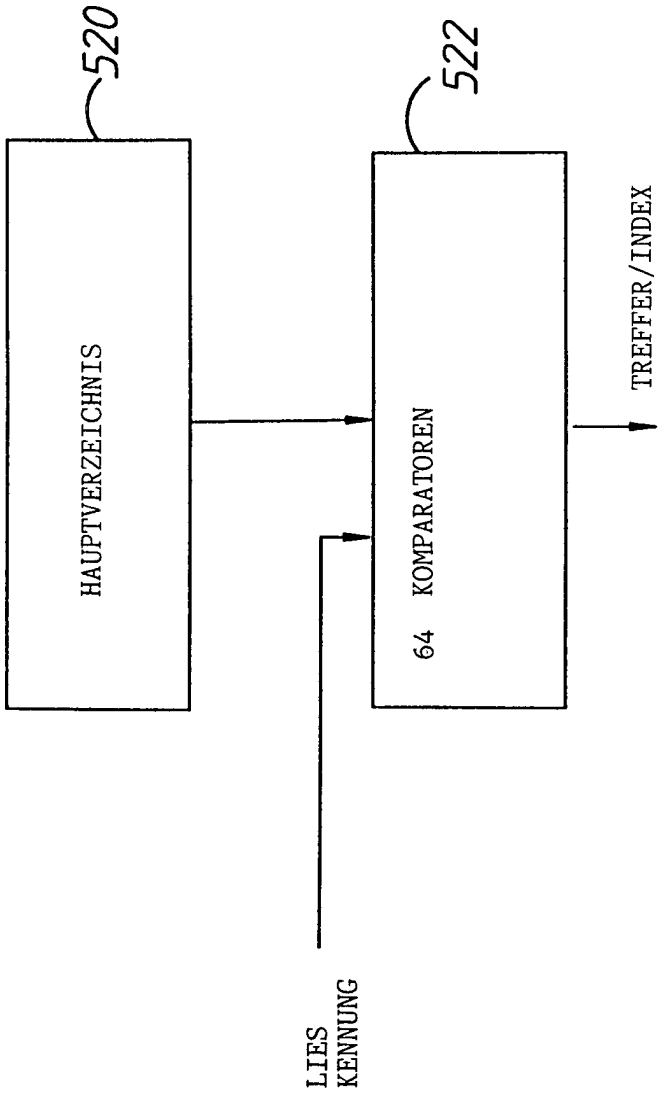

520
522
HAUPTVERZEICHNIS
64 KOMPARATOREN
TREFFER/INDEX
LIES KENNUNG

FIG 22

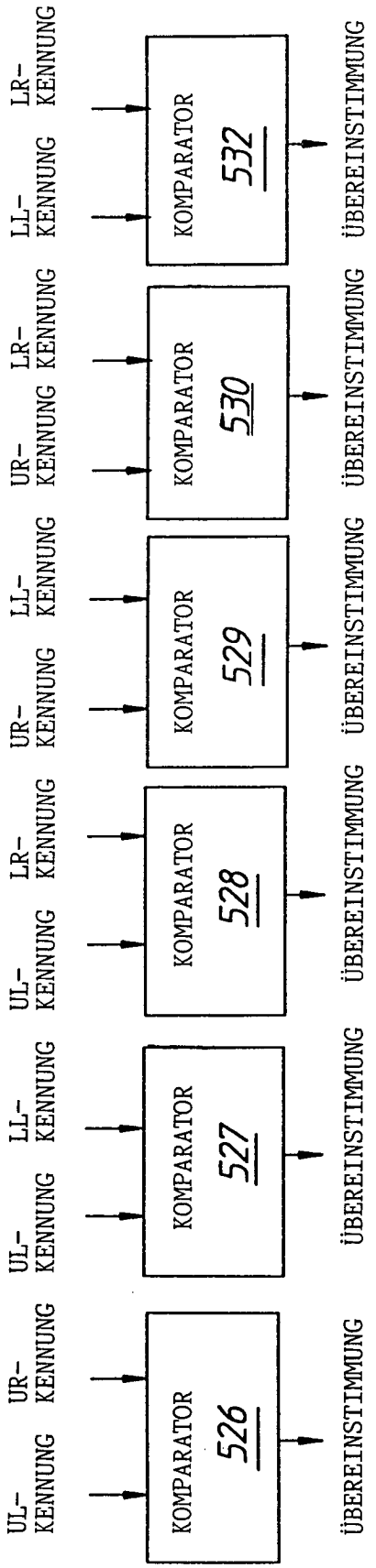

UL-KENNUNG
UR-KENNUNG
KOMPARATOR 526
ÜBEREINSTIMMUNG
UL-KENNUNG
LL-KENNUNG
KOMPARATOR 527
ÜBEREINSTIMMUNG
UL-KENNUNG
LR-KENNUNG
KOMPARATOR 528
ÜBEREINSTIMMUNG
UR-KENNUNG
LL-KENNUNG
KOMPARATOR 529
ÜBEREINSTIMMUNG
UR-KENNUNG
LR-KENNUNG
KOMPARATOR 530
ÜBEREINSTIMMUNG
LL-KENNUNG
LR-KENNUNG
KOMPARATOR 532
ÜBEREINSTIMMUNG

FIG 23

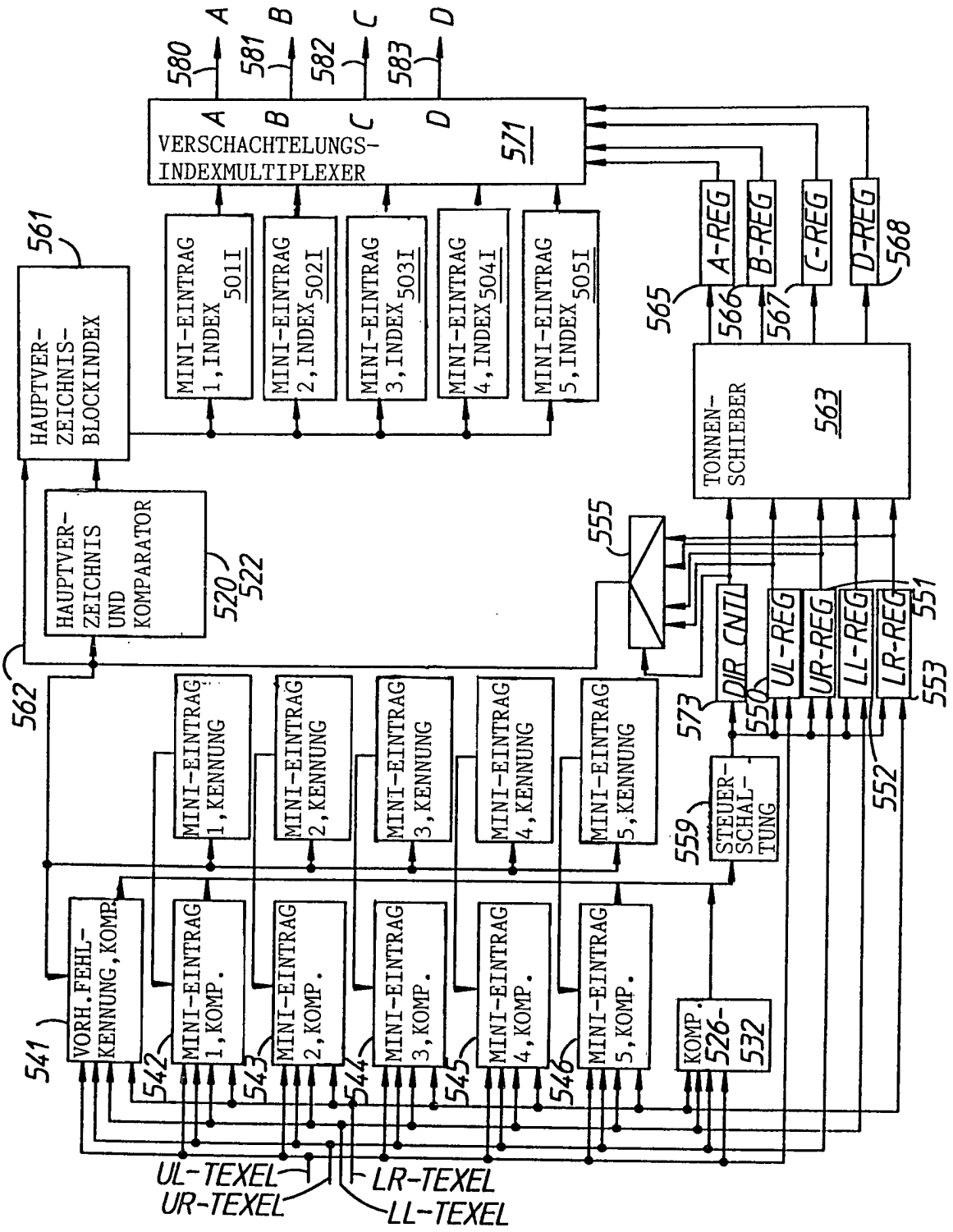

580
581
582
583
A
B
C
D
VERSCHACHTELUNGS-INDEXMULTIPLEXER
571
MINI-EINTRAG 1, INDEX 501I
MINI-EINTRAG 2, INDEX 502I
MINI-EINTRAG 3, INDEX 503I
MINI-EINTRAG 4, INDEX 504I
MINI-EINTRAG 5, INDEX 505I
561
HAUPTVER-ZEICHNIS-BLOCKINDEX
HAUPTVER-ZEICHNIS UND KOMPARATOR
520
522
562
A-REG
B-REG
C-REG
D-REG
565
566
567
568
TONNEN-SCHIEBER
563
555
DIR CNTL
UL-REG
UR-REG
LL-REG
LR-REG
551
553
573
550
552
559
STEUER-SCHAL-TUNG
MINI-EINTRAG 1, KENNUNG
MINI-EINTRAG 2, KENNUNG
MINI-EINTRAG 3, KENNUNG
MINI-EINTRAG 4, KENNUNG
MINI-EINTRAG 5, KENNUNG
VORH. FEHL-KENNUNG, KOMP.
MINI-EINTRAG 1, KOMP.
MINI-EINTRAG 2, KOMP.
MINI-EINTRAG 3, KOMP.
MINI-EINTRAG 4, KOMP.
MINI-EINTRAG 5, KOMP.
KOMP.
526-532
541
542
543
544
545
546
UL-TEXEL
UR-TEXEL
LR-TEXEL
LL-TEXEL

FIG 24