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

(10) **DE 197 09 227 B4** 2006.05.24

(12) **Patentschrift**

(21) Aktenzeichen: **197 09 227.6**
(22) Anmeldetag: **06.03.1997**
(43) Offenlegungstag: **29.01.1998**
(45) Veröffentlichungstag der Patenterteilung: **24.05.2006**

(51) Int Cl.[8]: ***G06T 1/60*** (2006.01)
G06T 15/00 (2006.01)

(30) Unionspriorität:
08/690,426 26.07.1996 US

(73) Patentinhaber:
Hewlett-Packard Development Co., L.P., Houston, Tex., US

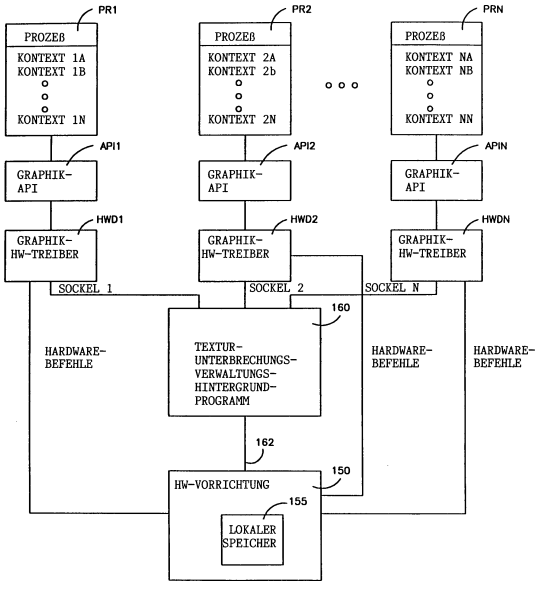
(74) Vertreter:
Schoppe, F., Dipl.-Ing.Univ., Pat.-Anw., 82049 Pullach

(72) Erfinder:
Cunniff, Ross, Fort Collins, Col., US; Saunders, Bradley L., Fort Collins, Col., US

(56) Für die Beurteilung der Patentfähigkeit in Betracht gezogene Druckschriften:
US 54 79 634 A
EP 000 62 165 A2
EP 07 49 100 A2
MESSMER, H.-P.: PC-Hardwarebuch, Addison-Wesley, 1993, S. 233-240;

(54) Bezeichnung: **Verfahren zum schnellen Herunterladen von Texturen auf eine Hardware für beschleunigte Graphiken und zur Beseitigung von zusätzlichen Softwarekopien von Texeln**

(57) Hauptanspruch: Verfahren zum Bereitstellen von Texturtabellen für die Erzeugung einer Graphik durch ein Computergraphiksystem, wobei das Computergraphiksystem einen Host-Computer mit zugeordnetem Systemspeicher und einer Graphikhardwarevorrichtung (150) zum Aufbereiten von mit einer Textur versehenen Bildern umfasst, wobei die Graphikhardwarevorrichtung (150) einen lokalen Speicher (155) aufweist und mit dem Host-Computer in Verbindung ist, und wobei der Host-Computer konfiguriert ist, um ein Anwendungsprogramm (PR1, PR2, PR3), eine Graphik-Anwendungsprogrammierschnittstelle (Graphik-API) (API1, API2, API3) und ein Hintergrundprogramm (160) auszuführen, wobei das Verfahren folgende Schritte aufweist:
(a) ansprechend auf einen gültigen Aufruf (52, 54) der Graphik-API (API1, API2, API3) durch das Anwendungsprogramm (PR1, PR2, PR3), Laden (56) von durch das Anwendungsprogramm (PR1, PR2, PR3) angegebenen Texturtabellen in den lokalen Speicher (155) der Graphikhardwarevorrichtung (150) ohne eine Kopie der Texturtabellen in dem Systemspeicher des Host-Computer zu erzeugen;
(b) Setzen (58) einer Flag, um die Existenz und die Gültigkeit der in den lokalen Speicher (155) der Graphikhardwarevorrichtung...



Beschreibung

[0001] Die vorliegende Erfindung bezieht sich auf ein Verfahren zum schnellen Herunterladen von Texturen auf eine Hardware für beschleunigte Graphiken und zur Beseitigung von zusätzlichen Softwarekopien von Texeln. Insbesondere bezieht sich die Erfindung auf eine Software-Speicherverwaltung von Texturtabellen eines Texturabbildungs-Computergraphiksystems und noch spezieller auf einen neuen Lösungsansatz, der das Herunterladen von Texturen beträchtlich beschleunigt, wenn ein Texturunterbrechungs-Verwaltungshintergrundprozeß ("TIM"-Daemon; TIM = Texture Interveniere Management), der auf einem virtuellen Speicherverwaltungssystem basiert, verwendet ist, um ein Hardwarevorrichtungs-Lokal-Cache-Speichersystem zum Speichern von Texturabbildungsdaten zu liefern.

Stand der Technik

[0002] Gegenwärtige Implementierungen einer Texturabbildung, die beispielsweise in der nachveröffentlichten EP 0 749 100 A2 mit dem Titel TEXEL CACHE INTERRUPT DAEMON FOR VIRTUAL MEMORY MANAGEMENT OF TEXTURE MAPS der Anmelderin der vorliegenden Anmeldung beschrieben sind, speichern eine Kopie der Textur des Benutzers softwaremäßig, um einen Mechanismus für eine Texturabfrage zu liefern und das hardwaremäßige Zwischenspeichern (Cacheinq) von Texeln zu ermöglichen, wenn nicht ausreichend Speicher existiert, daß alle Texel gleichzeitig in die Hardware passen.

[0003] Bei typischen Computergraphiksystemen ist ein Objekt, das auf dem Anzeigebildschirm dargestellt werden soll, in eine Mehrzahl von Graphik-Grundelementen unterteilt. Grundelemente sind elementare Komponenten eines Graphikbilds und können Punkte, Linien, Vektoren oder Polygone, beispielsweise Dreiecke, enthalten. Typischerweise ist ein Hardware/Software-Schema implementiert, um die Graphikgrundelemente, die die Ansicht von einem oder mehreren Objekten, das (die) auf dem Bildschirm dargestellt wird (werden), darstellen, für den zweidimensionalen Anzeigebildschirm aufzubereiten oder auf demselben 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 Form von x-, y-, z-Koordinaten der Scheitelpunkte desselben definieren, ebenso wie von R-, G-, B-Farbwerten jedes Scheitelpunkts. Eine Aufbereitungshardware interpoliert die Grundelementdaten, um die Anzeigebildschirm- pixel zu berechnen, die eingeschaltet werden, um jedes Grundelement darzustellen, ebenso wie die R-, G-, B-Werte für jedes Pixel.

[0005] Frühe Graphiksysteme 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 solche Systeme angezeigt wurden, zeigten extrem glatte Oberflächen ohne Texturen, Erhebungen, Vertiefungen, Schatten oder anderen Oberflächendetails, die bei dem Objekt, das modelliert wird, vorliegen.

[0006] Folglich wurden Verfahren entwickelt, um Bilder mit verbesserten Oberflächeneinzelheiten anzuzeigen. Die Texturabbildung ist ein solches Verfahren, das das Abbilden eines Quellenbilds, das hierin als eine "Textur" bezeichnet wird, auf eine Oberfläche eines dreidimensionalen Objekts und nachfolgend das Abbilden des texturierten dreidimensionalen Objekts auf dem zweidimensionalen Graphikanzeigebildschirm, um das resultierende Bild anzuzeigen, enthält. Oberflächen-Detailattribute, die üblicherweise mittels der Texturabbildung abgebildet werden, umfassen die Farbe, Spiegelreflexionen, Vektorperturbationen, das Spiegelvermögen, die Transparenz, Schatten, Oberflächen-Unregelmäßigkeiten sowie Abstufungen.

[0007] Die Texturabbildung umfaßt das Anwenden von einem oder mehreren Punkttexturelementen ("Texeln") auf jedes Punktelement ("Pixel") des angezeigten Abschnitt des Objekts, auf das die Textur abgebildet wird. Die Texturabbildungshardware wird üblicherweise mit Informationen beliefert, 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 Ort desselben in der zweidimensionalen Texturtabelle identifizieren. Für jedes Pixel wird aus der Texturtabelle auf das entsprechende Texel oder die Texel, die auf dasselbe abbilden, zugegriffen, wobei dieselben in die endgültigen R-, G-, B-Werte, die für das Pixel erzeugt werden, eingearbeitet werden, um das texturierte Objekt auf dem Anzeigebildschirm darzustellen.

[0008] Es sollte offensichtlich sein, daß jedes Pixel in einem Objektgrundelement für jede Ansicht des Objekts nicht in einer Eins-zu-Eins-Entsprechung auf ein einzelnes Texel in der Texturtabelle abbildet. Beispielsweise wird, je näher das Objekt der Sichtebene, die auf dem Anzeigebildschirm dargestellt ist, ist, das Objekt umso größer erscheinen. Wenn das Objekt auf dem Anzeigebildschirm größer erscheint, wird die Darstellung der Textur detaillierter. Wenn das Objekt einen ziemlich großen Abschnitt des Anzeigebildschirms belegt, wird somit eine große Anzahl von Pixeln verwendet, um das Objekt auf dem Anzeigebild-

schirm darzustellen, wobei jedes Pixel, das das Objekt darstellt, in einer Eins-zu-Eins-Entsprechung auf ein einzelnes Texel in der Texturtabelle abbilden kann, wobei alternativ ein einzelnes Texel auf mehrere Pixel abbilden kann. Wenn das Objekt jedoch einen relativ kleinen Abschnitt auf dem Anzeigebildschirm besetzt, ist eine viel kleinere Anzahl von Pixeln verwendet, um das Objekt darzustellen, was zur Folge hat, daß die Textur weniger detailliert dargestellt wird, so daß jedes Pixel auf eine Mehrzahl von Texeln abbilden kann. Außerdem kann jedes Pixel in eine Mehrzahl von Texeln abbilden, wenn eine Textur auf einen kleinen Abschnitt eines Objekts abgebildet wird. Für jedes Pixel, das auf mehr als ein Texel abbildet, werden resultierende Texeldaten berechnet. Da es üblich ist, daß ein Pixel auf mehrere Texel abbildet, stellen resultierende Texeldaten für ein Pixel typischerweise einen Mittelwert der Texel, die auf dieses Pixel abbilden, dar.

[0009] Texturabbildungs-Hardwaresysteme weisen typischerweise einen lokalen Speicher auf, der Daten speichert, die eine Textur darstellen, die dem Objekt, das aufbereitet wird, zugeordnet ist. Wie oben erläutert wurde, kann ein Pixel auf mehrere Texel abbilden. Wenn es notwendig wäre, daß die Texturabbildungshardware eine große Anzahl von Texeln, die auf ein Pixel abbilden, aus dem Lokalspeicher liest, um einen Mittelwert zu erzeugen, dann wäre eine große Anzahl von Speicher-Auslesungen und die Mittelwertbildung von vielen Texelwerten erforderlich, was zeitaufwendig wäre und das Systemverhalten verschlechtern würde.

[0010] Um dieses Problem zu lösen, wurde ein Schema entwickelt, das die Erzeugung einer Reihe von Tabellen, die "MIP"-Tabellen ("MIP" bedeutet "multum in parvo" = viele Dinge an einem kleinen Ort) genannt werden, für jede Textur enthält, und das Speichern der MIP-Tabellen der Textur, die dem Objekt, das aufbereitet wird, zugeordnet ist, in dem Lokalspeicher der Texturabbildungshardware. Eine MIP-Tabelle für eine Textur weist eine Basistabelle auf, die direkt der Texturtabelle entspricht, ebenso wie eine Reihe von gefilterten Tabellen, in denen jede aufeinanderfolgende Tabelle größenmäßig um einen Faktor von zwei in jeder der zwei Texturtabellendimensionen reduziert ist. Ein darstellendes Beispiel eines Satzes von MIP-Tabellen ist in **Fig.** 1 gezeigt. Die MIP-Tabellen weisen eine Basistabelle ("Ebene 0") **100** auf, die acht mal acht Texel groß ist, ebenso wie eine Reihe von Tabellen **102**, **104** und **108**, die eine Ebene 1, die vier mal vier Texel groß ist, eine Ebene 2, die zwei mal zwei Texel groß ist, bzw. eine Ebene 3, die 1 Texel groß ist, darstellen.

[0011] Die Ebene-1-Tabelle **102** der Größe vier mal vier wird durch eine Blockfilterung (Dezimierung) der Basistabelle **100** erzeugt, derart, daß jedes Texel in der Ebene-1-Tabelle **102** einem Mittelwert von vier Texeln aus der Ebene-0-Basistabelle **100** entspricht. Beispielsweise ist das Texel **110** in der Ebene-1-Tabelle **102** gleich dem Mittelwert der Texel **112** bis **115** der Ebene-0-(Basis-)Tabelle **100**. In gleicher Weise sind die Texel **118** und **120** in der Ebene-1-Tabelle **102** gleich den Mittelwerten der Texel **121** bis **124** bzw. **125** bis **128** der Ebene-0-(Basis-)Tabelle **100**. Die Ebene-2-Tabelle **104** der Größe zwei mal zwei ist in gleicher Weise durch eine Blockfilterung der Ebene-1-Tabelle **102** gebildet, derart, daß das Texel **130** in der Ebene-2-Tabelle **104** gleich dem Mittelwert der Texel **110** und **118** bis **120** der Ebene-1-Tabelle **102** ist. Das einzelne Texel in der Ebene-3-Tabelle **108** ist durch die Mittelwertbildung der vier Texel in der Ebene-2-Tabelle **104** erzeugt.

[0012] Herkömmliche Graphiksysteme laden üblicherweise die gesamte Reihe von MIP-Tabellen für jede Textur, die für die Grundelemente, die auf dem Anzeigebildschirm gezeichnet werden sollen, verwendet werden soll, aus dem Hauptspeicher des Hostcomputers in den Lokalspeicher der Texturabbildungshardware. Folglich kann die Texturabbildungshardware auf Texturdaten aus jeder der Reihe von MIP-Tabellen zugreifen. Die Bestimmung dessen, auf welche Tabelle zuzugreifen ist, um die Texeldaten für jedes spezielle Pixel zu liefern, basiert auf der Anzahl von Texeln, die dem Pixel zugeordnet ist. Wenn dem Pixel ein einzelnes Texel in der Texturtabelle in einer Eins-zu-Eins-Entsprechung zugeordnet ist, wird beispielsweise auf die Basistabelle **100** zugegriffen. Wenn das Pixel jedoch auf vier, sechzehn oder vierundsechzig Texel abbildet, wird jeweils auf die Tabellen **102**, **104** und **108** zugegriffen, da diese Tabellen jeweils Texeldaten speichern, die einen Mittelwert von vier, sechzehn und vierundsechzig Texeln in der Texturtabelle speichern.

[0013] Überdies sind einige Texturabbildungssysteme pipelineartig aufgebaut, so daß verschiedene Operationen auf unterschiedlichen Objektgrundelementen gleichzeitig 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 auf einmal zu speichern. Wenn ein Wechseln der Textur, die bei der Aufbereitung von Grundelementen verwendet ist, vorliegt, muß das System folglich eine neue Reihe von MIP-Tabellen herunterladen. Typischerweise umfaßt der Datenweg, der verwendet ist, um die neuen Texturdaten in den Lokalspeicher der Texturabbildungshardware herunterzuladen, das Durchlaufen der Grundelement-Aufbereitungspipeline des Systems. Wenn eine neue Textur abgebildet werden soll, muß daher ermöglicht sein, daß die Grundelement-Aufbereitungspipeline entleert ist, bevor die neue Reihe von MIP-Tabellen heruntergeladen werden kann. Sobald die Reihe von MIP-Tabellen heruntergeladen ist, muß die Pipeline wieder gefüllt werden. Die Notwen-

digkeit des Spülens der Grundelement-Aufbereitungspipeline, jedesmal, wenn eine neue Textur erforderlich ist, reduziert die Bandbreite des Systems.

[0014] **Fig.** 2 ist ein Blockdiagramm, teils Hardware, teils hierarchische Software, eines herkömmlichen Computergraphiksystems, das die Softwareschichten zeigt, die auf dem Prozessor des Systemhostcomputers ablaufen, ebenso wie die Hardwarevorrichtung, mit der der Hostcomputer kommuniziert, um die mit einer Textur versehenen Bilder aufzubereiten. Der Systemspeicher des Hostcomputers kann eine Mehrzahl Benutzerinteraktiver Prozesse PR1, PR2, ..., PRN speichern, die auf dem Systemprozessor laufen können, wie in **Fig.** 2 gezeigt ist. Jeder Prozeß, PRi, ist ein Hochpegel-Computergraphik-Softwareprogramm oder eine Anwendung, beispielsweise eine Datenbank, eine CAD/CAM-Anwendung, eine Architekturentwurfsanwendung, eine Bautechnikanwendung, ein Textverarbeitungs-Datenpaket oder dergleichen.

[0015] Innerhalb eines speziellen Prozesses kann ein Benutzer mehrere Kontexte erzeugen. Jeder Kontext, der in einem Prozeß erzeugt wird, kann eine unterschiedliche Ansicht des gleichen Bilds enthalten. Beispielsweise kann ein Benutzer in einem Bautechnik-Entwurfsanwendungsprozeß unterschiedliche Ansichten der gleichen Struktur, beispielsweise eines Gebäudes, erzeugen. Jeder Kontext kann eine Texturabbildung erfordern, um das Bild aufzubereiten. Bei diesem Beispiel können Texturtabellen erforderlich sein, um die Böden, Wände, Decken und andere Oberflächenattribute des Gebäudes zu zeichnen. Folglich liefert der Benutzer die Textur-Tabelle oder -Tabellen, um das Bild dieses Kontextes aufzubereiten. Da mehrere Kontexte, die in dem gleichen Prozeß erzeugt werden, typischerweise unterschiedliche Ansichten des gleichen Bilds umfassen, benötigen die Kontexte innerhalb eines Prozesses üblicherweise die gleiche Textur oder die gleichen Texturen.

[0016] Wenn ein Kontext durch einen Benutzer erzeugt wird, erteilt der Benutzer einen Befehl, der für die Hochpegel-Prozeßprogrammiersprache erkennbar ist, um das Bild zu zeichnen. Wenn das Bild eine Texturtabelle erfordert, wird ein Befehl eingegeben, der anzeigt, daß eine Texturtabelle erforderlich ist, wobei die Texturtabelle dann durch den Benutzer von einer externen Eingabevorrichtung, einer Floppy-Diskette oder dergleichen, geliefert wird. Der Prozeß, oder die Hochpegel-Programmiersprache, übersetzt dann diesen Befehl in einen Niederpegel-Softwarebefehl und kopiert die Textur, die durch den Benutzer eingegeben wird. Bei bekannten Systemen, beispielsweise dem, das in **Fig.** 2 gezeigt ist, erfordert jeder Kontext, der in einem einzelnen Prozeß erzeugt wird, seine eigene Kopie dieser Textur. Jeder Kontext speichert seine Kopie der Textur in einem zugewiesenen Bereich des Systemspeichers auf dem Hostcomputer. Folglich werden, wenn mehrere Kontexte innerhalb eines Prozesses die Verwendung der gleichen Textur erfordern, mehrere Kopien dieser Textur in dem Systemspeicher gespeichert, wobei eine Kopie jedem Kontext zugeordnet ist.

[0017] Der vom Benutzer eingegebene Befehl, um ein Bild unter Verwendung einer speziellen Textur zu zeichnen, wird, sobald er durch den Prozeß übersetzt ist, von dem Prozeß zu einer darunterliegenden Graphikanwendungsprogrammier-Schnittstelle ("API"; API = Application Programmer Interface) übertragen. Wie in **Fig.** 2 gezeigt ist, kommuniziert eine eindeutige Graphik-API mit jedem Prozeß des Systems. Jede Graphik-API ist eine darunterliegende Niederpegel-Softwaresprache, die auf dem Prozessor des Hostcomputers arbeitet, um jeden Hochpegel-Graphikbefehl, der von dem entsprechenden Prozeß empfangen wird, in einen Niederpegel-Softwarecodebefehl zu übersetzen. Die Graphik-API speichert ferner eine Kopie jeder Textur, die erforderlich ist, um das Bild aufzubereiten, in ihrem eigenen zugewiesenen Ort des Systemspeichers. Die Graphik-API unterteilt das Bild, das aufbereitet werden soll, zusätzlich in Graphikkomponenten, beispielsweise Vierecke oder Polygone. Derartige Informationen werden dann einem entsprechenden Hardwaretreiber geliefert.

[0018] Ein eindeutiger Hardwaretreiber kommuniziert mit jeder Graphik-API. Jeder Hardwaretreiber ist eine weitere darunterliegende Graphiksoftwaresprache niedrigeren Pegels, die jeden Niederpegel-Graphikbefehl, der von der API empfangen wird, in Hardwarebefehle übersetzt, die für die Graphikhardwarevorrichtung, mit der der Hostcomputer verbunden ist, erkennbar ist. Diese Befehle werden dann von dem Hardwaretreiber zu der Hardwarevorrichtung übertragen, die als Reaktion das mit der Textur versehene Bild auf einen Anzeigebildschirm, der mit der Hardwarevorrichtung verbunden ist, zeichnet. Wie in **Fig.** 2 gezeigt ist, kommuniziert jedes Beispiel eines Hardwaretreibers HWD1, HWD2, ..., HWDN mit einer jeweiligen Graphik-API, API1, API2, ..., APIN.

[0019] Bei dem Beispiel, das in **Fig.** 2 gezeigt ist, sind alle Hardwaretreiber mit einer einzelnen Graphikhardwarevorrichtung **150** verbunden. Es ist für Fachleute jedoch offensichtlich, daß der Hostcomputer mit einer Mehrzahl unterschiedlicher Graphikhardwarevorrichtungen verbunden sein kann, wobei jeweils ein unterschiedlicher Hardwaretreiber mit jeder Hardwarevorrichtung kommuniziert.

[0020] Wenn eine Graphik-API einen Niederpegel-Softwarebefehl liefert, um ein mit einer Textur versehenes Polygon unter Verwendung einer speziellen Textur, die ebenfalls geliefert wird, aufzubereiten, übersetzt der Graphikhardwaretreiber den Niederpegelbefehl in Hardwarebefehle, die für die Hardware-

vorrichtung **150** erkennbar sind. Der Graphikhardwaretreiber liefert danach die Befehle, um das Polygon aufzubereiten, zu der Hardwarevorrichtung **150**, indem dieselben in geeignete Register und Eingangspuffer in der Hardwarevorrichtung geschrieben werden. Zusätzlich liefert der Graphikhardwaretreiber die Texturdaten für dieses spezielle Polygon zu dem Lokalspeicher **155** der Hardwarevorrichtung **150**, wo dieselben während der Aufbereitung dieses Polygons temporär gespeichert werden. Wie hierin beschrieben ist, laden bekannte Computergraphiksysteme die gesamte Reihe von MIP-Tabellen für eine einzelne Textur aus dem Graphikhardwaretreiber in den Lokalspeicher der Hardwarevorrichtung herunter, jedesmal, wenn diese Textur erforderlich ist, um ein Bild oder eine Komponente des Bilds aufzubereiten.

[0021] Bei herkömmlichen Systemen, beispielsweise dem, das in **Fig.** 2 gezeigt ist, werden mehrere Kopien einer einzelnen Textur an unterschiedlichen Orten des Systemsoftwarespeichers des Hostcomputers gespeichert. Wenn beispielsweise der Kontext **1A** und der Kontext **1B** des Prozesses PR1 die gleiche Textur erfordern, um unterschiedliche Ansichten eines speziellen Bilds aufzubereiten, liefert der Benutzer eine erste Kopie dieser Textur, wenn dieselbe anfänglich definiert und in den Prozeß eingeführt wird. Die Graphik-API1 erstellt eine Kopie der Textur für den Kontext **1A** (die zweite Kopie) und eine weitere Kopie dieser Textur für den Kontext **1B** (die dritte Kopie) und speichert dieselben in einem Speicher, der von der API und der Hardware gemeinsam verwendet wird. Folglich sind drei einzelne Kopien der gleichen Textur an unterschiedlichen Orten des Systemsoftwarespeichers gespeichert, um zwei Ansichten des gleichen Bilds unter Verwendung der gleichen Textur aufzubereiten. Zusätzlich werden eine vierte und eine fünfte Kopie der Textur der Hardwarevorrichtung geliefert und in dem Lokalspeicher derselben gespeichert.

[0022] Wie offensichtlich wird, kann eine Reihe von Textur-MIP-Tabellen zur Speicherung eine große Systemsoftwarespeichermenge erfordern. Beispielsweise erfordert eine Reihe von MIP-Tabellen für eine Textur mit einer Texturbasistabelle von 1.024 × 1.024 Texeln mehr als 5 MByte Systemsoftwarespeicher, um eine Kopie der MIP-abgebildeten Textur zu speichern. Folglich verbraucht die Mehrzahl gespeicherter Kopien der MIP-abgebildeten Textur einen signifikanten Systemsoftwarespeicherbetrag.

[0023] Obwohl Systemsoftwarespeicher in der Lage sein können, Softwaredaten bis zu einigen Gigabyte zu speichern, können typische, zweckgebundene, lokale Hardwaretexturspeicher einer Texturhardwarevorrichtung viel weniger Daten speichern. Derartige lokale Texturhardwarespeicher liegen typischerweise in einem Bereich von vier MByte bis sechzehn MByte. In vielen Texturabbildungsanwendungen übersteigt daher der Systemspeicherbetrag, der durch Texturen verbraucht wird, den des lokalen Hardwaretexturspeichers weit. Obwohl der Systemspeicher typischerweise mehrere MIP-abgebildete Texturen für die Verwendung mit mehreren Kontexten eines Prozesses speichert, kann der lokale Hardwarevorrichtungsspeicher gleichzeitig nur eine Textur einer begrenzten Größe speichern. Daher ist die ordnungsgemäße Verwaltung des Lokaltexturspeichers der Hardwarevorrichtung kritisch, um ein maximales Verhalten des Systems zu erreichen.

[0024] Zusätzlich zu dem großen Systemsoftwarespeicherverbrauch aufgrund der Speicherung mehrerer Kopien von Texturen leidet die Systembandbreite allgemein unter herkömmlichen lokalen Hardwaretexturspeicher-Verwaltungsschemata. Herkömmliche lokale Hardwarespeicher-Verwaltungsschemata ersetzen wiederholt ganze Reihen von Textur-MIP-Tabellen in dem lokalen Hardwarespeicher. Die Reihe von MIP-Tabellen für eine Textur wird in dem lokalen Speicher jedesmal ersetzt, wenn diese Textur benötigt wird, um ein Polygon aufzubereiten. Derartige Schemata sind weder in der Lage, die Geschichte der Verwendung dieser Textur zu berücksichtigen, noch die zukünftige Verwendung dieser Textur vorherzusagen. Durch das wiederholte Herunterladen der gesamten Reihe der gleichen Textur von dem Systemspeicher in den lokalen Hardwaretexturspeicher wird die Systembandbreite negativ beeinflußt.

[0025] Obwohl herkömmliche Texturhardwarespeicher-Ersetzungsalgorithmen ganze Textur-MIP-Tabellen-Reihen herunterladen, darf jede Textur-MIP-Tabellen-Reihe die physikalischen Speicherfähigkeitsgrenzen des lokalen Hardwaretexturspeichers nicht überschreiten. Daher muß der Benutzer des Graphikprozesses, der mit der speziellen Hardwarevorrichtung verbunden ist, die Kapazität des Lokaltexturspeichers kennen, so daß keine Texturen verwendet werden, die größer sind als diese Kapazität. Obwohl viele Anwendungen (Prozesse) mit mehreren, unterschiedlichen, darunterliegenden Hardwarevorrichtungen arbeiten können, von denen jede eindeutige Lokaltexturspeicher-Beschränkungen aufweist, fällt die Last dem Benutzer zu, Kenntnis von solchen Beschränkungen zu besitzen und Bilder auf eine vorrichtungsabhängige Art und Weise zu erzeugen.

[0026] Wie oben angezeigt wurde, ist es möglich, daß für jede gegebene Textur eine große Anzahl von Softwarekopien erzeugt werden muß. Jedoch existiert in vielen Fällen niemals ein Bedarf nach all diesen Softwarekopien. In diesen Fällen fragt der Benutzer die Textur niemals wieder ab, wobei während der Lebensdauer der Textur stets Raum benötigt wird, um dieselbe in dem Texel-Cache zu speichern. Frü-

here Lösungen für dieses Problem richteten sich auf das Reduzieren der Anzahl von Operationen, die bezüglich der Texturtabelle durchgeführt wurden, bevor dieselbe in den Graphikkernspeicher kopiert wurde, wobei dieselben jedoch nicht die verschiedenen Softwarekopien beseitigten, da davon ausgegangen wurde, daß die Softwarekopien benötigt werden würden, da der Graphikkern zukünftige Funktionen, die auf der Textur durchgeführt werden könnten (beispielsweise eine Abfrage der Textur), nicht bestimmen kann.

[0027] Entsprechend ist in der EP 0 749 100 A2 1995 mit dem Titel TEXEL CACHE INTERRUPT DAEMON FOR VIRTUAL MEMORY MANAGEMENT OF TEXTURE MAPS, auf die oben Bezug genommen wurde, ein TIM beschrieben, der die gemeinsame Speicherung von Texturen unter dem Verwalter (dem "TIM"), dem Hardwaretreiber und der entsprechenden Graphik-API in dem Systemspeicher liefert. Der Verwalter liefert ferner eine gemeinsame Speicherung von Texturen unter mehreren Kontexten eines einzelnen Prozesses. Das Verwaltungsschema, das von Gannett beschrieben ist, ermöglicht ferner, daß der Benutzer der Hochpegel-Graphikprozesse Bilder auf eine vorrichtungsunabhängige Art und Weise erzeugt, ohne Kenntnis der Speicherfähigkeitsgrenzen der Lokalspeicher der darunterliegenden Hardwarevorrichtungen.

[0028] Ein Ausführungsbeispiel der Erfindung von Gannett ist in Blockdiagrammform in **Fig.** 3 gezeigt, bei der zur Bezeichnung von Elementen, die identisch zu denen von **Fig.** 2 sind, gleiche Bezugszeichen verwendet sind. Wie bei dem bekannten Ausführungsbeispiel von **Fig.** 2 (vor Gannett) weist das System mehrere Benutzer-interaktive Prozesse PR1, PR2, ..., PRN auf, die auf dem Systemprozessor laufen. In jedem Prozeß kann der Benutzer mehrere Kontexte erzeugen, die die gleichen Texturtabellen benötigen. Das System weist ferner eine darunterliegende Graphik-API und einen Graphikhardwaretreiber für jeden Prozeß auf.

[0029] Die Erfindung von Gannett bezog sich auf ein Texturunterbrechungsverwaltungs-Hintergrundprogramm ("TIM"-Daemon) **160**, das ein unabhängiger alleinstehender Softwareprozeß ist, der auf dem Prozessor des Hostcomputers ohne Kenntnis des Benutzers abläuft. Der TIM **160** der Erfindung von Gannett steht mit jedem der Graphikhardwaretreiber über einen unterschiedlichen Sockel in Verbindung. Wie für Fachleute offensichtlich ist, ist ein Sockel ein Softwarekommunikationsweg, über den zwei Softwareprogramme kommunizieren können. Der TIM kommuniziert mit der Hardwarevorrichtung **150** ferner über einen Bus **162**.

[0030] Der TIM der Erfindung von Gannett kann auf herkömmlichen Prozessoren von Computergraphik-Systemhostcomputern, die als Computergraphikworkstations bekannt sind, ablaufen. Ein Beispiel einer derartigen Workstation ist die Hewlett-Packard-9000-Reihe J200.

[0031] Der TIM **160** verwaltet das Speichern von Texturdaten in dem Lokalspeicher **155** der Hardwarevorrichtung **150**, wobei der TIM einen vorrichtungsunabhängigen Abschnitt enthielt, der die Softwaretexturspeicher-Verwaltung handhabte und sockelmäßige Verbindungen mit den Graphikhardwaretreibern und einem vorrichtungsabhängigen Abschnitt, der über den Bus **162** in die Hardwarevorrichtung schreibt und von derselben liest.

[0032] Bei einem Ausführungsbeispiel der Erfindung von Gannett ist der Lokalspeicher der Hardwarevorrichtung als ein Cache-Speicher angeordnet, bei dem Abschnitte von Texturen zu jeder einzelnen Zeit in dem Lokalspeicher der Hardwarevorrichtung gespeichert sind. Die vorrichtungsunabhängigen Abschnitte des TIM verfolgen die Verwendung der Abschnitte von Texturdaten, die in dem Cache gespeichert sind und überwachen die Prioritäten der Texturen, um die zukünftige Verwendung dieser Abschnitte vorherzusagen. Ein Cache-Fehlgriff in der Hardware findet statt, wenn von der Hardwarevorrichtung Texturdaten benötigt werden, um ein Bild aufzubereiten, die gegenwärtig nicht in dem Cache gespeichert sind. Wenn ein Cache-Fehlgriff stattfindet, bestimmt der TIM, welcher Block von Texturdaten in dem Cache zu ersetzen ist, indem berücksichtigt wird, welcher Block oder welche Blöcke von Texturdaten in dem Cache zuletzt verwendet wurden, und welche Texturen die geringste Priorität aufweisen.

[0033] Bei einem Ausführungsbeispiel der Erfindung von Gannett weist der TIM einen gemeinsam verwendeten Speicherort in dem Systemspeicher auf, an dem eine große Textur oder ein Texturabschnitt gespeichert ist, und zwischen der Graphik-API, dem Graphikhardwaretreiber und dem TIM für einen speziellen Prozeß gemeinsam verwendet wird. Sowohl die Graphik-API als auch der Graphikhardwaretreiber (für einen speziellen Prozeß) als auch der TIM **160** weisen einen Zeiger auf einen Ort in dem gemeinsam verwendeten Systemspeicherbereich auf, um auf die gespeicherte Texturkopie zuzugreifen. Zusätzlich liefert der TIM ein Speicherverwaltungsschema derart, daß alle Kontexte, die in einem einzelnen Prozeß erzeugt werden, die gleiche Kopie jeder Textur, die für diese Kontexte benötigt wird, gemeinsam verwenden. Für bestimmte Anwendungen liefert der TIM der Erfindung von Gannett daher enorme Einsparungen an Systemsoftwarespeicherraum.

[0034] Bezüglich des Beispiels, das oben bezugnehmend auf **Fig.** 2 beschrieben wurde, bei dem der Kontext **1A** und der Kontext **1B** des Prozesses PR1 die gleiche Textur benötigen, um unterschiedliche

Ansichten eines speziellen Bilds aufzubereiten, lieferte der TIM der Erfindung von Gannett enorme Systemspeicherraumeinsparungen, wenn die Textur groß ist. Im Gegensatz zu dem bekannten beschriebenen System (vor Gannett) kann eine einzelne Kopie der Textur gespeichert werden, die zwischen dem Kontext **1A** und dem Kontext **1B** gemeinsam verwendet wird. Wenn die Textur groß genug war, um zu gewährleisten, daß dieselbe an dem gemeinsam verwendeten Speicherort gespeichert ist, wird zusätzlich nur eine weitere Kopie der Textur an dem gemeinsam verwendeten Systemspeicherort gespeichert und zwischen dem TIM, der Graphik-API1 und dem Graphikhardwaretreiber HWD1 gemeinsam verwendet werden. Folglich werden bei diesem Beispiel nur drei einzelne Kopien der gleichen Textur an unterschiedlichen Orten des Systemsoftwarespeichers gespeichert, um zwei Ansichten des gleichen Bilds unter Verwendung der gleichen Textur aufzubereiten. Eine Kopie ist die Benutzerkopie, eine Kopie wird zwischen den zwei Kontexten des Prozesses PR1 gemeinsam verwendet, und eine Kopie wird zwischen der Graphik-API, dem Graphikhardwaretreiber und dem TIM gemeinsam verwendet. Bei dem bekannten System von **Fig.** 2 (vor Gannett), das oben beschrieben ist, hätten fünf Kopien dieser gleichen Textur in dem Systemsoftwarespeicher gespeichert werden müssen. Daher müssen bei dem TIM der Erfindung von Gannett zwei Kopien der Textur weniger in dem Systemsoftwarespeicher gespeichert werden, was für eine große Textur große Speicherraumeinsparungen zur Folge hat.

[0035] Andererseits erfordert die Verwendung der Vorrichtung der vorliegenden Erfindung, wie hierin nachfolgend erklärt wird, daß nur eine einzelne Kopie der Textur existiert, und daß dieselbe in der Graphikhardware beibehalten wird.

[0036] Unter Verwendung der Erfindung von Gannett war es notwendig, mehrere Schritte durchzuführen, um ein mit einer Textur versehenes Dreieck zu zeichnen. Wie in **Fig.** 4 gezeigt ist, spezifizierte das Anwendungsprogramm zuerst unter Verwendung eines API-Aufrufs eine Textur **12**. Die API-Bibliothek verifizierte, daß der API-Aufruf gültig war **14**, woraufhin die API die Textur in einen privaten Bibliothekspuffer kopierte **16**, wodurch die erste Kopie der Textur erzeugt wurde.

[0037] Als nächstes kopierte das Texturhintergrundprogramm ("TIM") die Textur in den privaten Puffer des Hintergrundprogramms (möglicherweise unformatiert, um Hintergrundprogrammanforderungen zu erfüllen), wobei die zweite Kopie der Textur erzeugt wird **18**.

[0038] Ein Anwendungsprogramm zeichnet ein Dreieck unter Verwendung der Textur, indem die Graphikhardware das Texturhintergrundprogramm auffordert, Abschnitte der Textur, die erforderlich sind, um das Dreieck zu zeichnen, zu liefern. Folglich muß das Texturhintergrundprogramm die erforderlichen Abschnitte der Textur für das Anwendungsprogramm herunterladen, wodurch die dritte Kopie der Textur erstellt wird **24**.

[0039] In ähnlicher Weise wird entsprechend der Erfindung von Gannett eine zusätzliche Kopie der Textur beibehalten, um die Abfrage durch die Anwendung des momentanen Werts der Textur zu handhaben, oder um die Beseitigung der Textur aus der Graphikhardware zu handhaben, wenn eine neue Textur benötigt wird.

Aufgabenstellung

[0040] Ausgehend von dem genannten Stand der Technik besteht die Aufgabe der vorliegenden Erfindung darin, ein Verfahren zum schnellen Herunterladen von Texturen in einem Computergraphik-Hardwaresystem zu schaffen, das nur eine geringe Anzahl von Kopien der Textur benötigt.

[0041] Diese Aufgabe wird durch ein Verfahren gemäß Anspruch 1 gelöst.

[0042] Gemäß dem bevorzugten Ausführungsbeispiel der Erfindung wird das Erstellen der Softwarekopie der Textur verzögert, bis dieselbe absolut benötigt wird oder die Textur durch den Benutzer verworfen wird.

[0043] Ferner verallgemeinert die vorliegende Erfindung ein Texturherunterladen, um stets diesen Mechanismus zu verwenden, was die Softwarearchitektur stark vereinfacht und das Verhalten in Fällen wie z.B. Rahmenpufferkopien in den Textur-Cache verbessert, wobei wiederum keine Softwarekopie benötigt wird, es sei denn, eine solche wird angefordert.

[0044] Das schnelle Herunterladen von Texturen zu einer Hardware für beschleunigte Graphiken, das einfach als "schnelles Herunterladen" bezeichnet wird, liefert einen Mechanismus, um Texturen direkt in den Hardware-Texel-Cache herunterzuladen, wobei der Bedarf nach einer Softwarekopie der Texel verzögert oder möglicherweise vollständig beseitigt wird. Dies verbessert das Verhalten des Herunterladens von Texturen in die Hardware dramatisch, was ferner das Verhalten von Anwendungen, die sich dynamisch ändernde Texturen aufweisen, stark verbessert.

Ausführungsbeispiel

[0045] Bevorzugte Ausführungsbeispiele der vorliegenden Erfindung werden nachfolgend bezugnehmend auf die beiliegenden Zeichnungen näher erläutert. Es zeigen:

[0046] Fig. 1 eine graphische Darstellung eines Satzes von Textur-MIP-Tabellen;

[0047] Fig. 2 ein Blockdiagramm, teils Hardware, teils hierarchische Software, eines bekannten Computergraphiksystems;

[0048] Fig. 3 ein Blockdiagramm, teils Hardware, teils Software, eines Computergraphiksystems gemäß der bekannten Erfindung von Gannett;

[0049] Fig. 4 ein Flußdiagramm, das das Verfahren, das durch die bekannte Erfindung von Gannett verwendet wird, darstellt; und

[0050] Fig. 5 und Fig. 6 Flußdiagramme, die das Verfahren der vorliegenden Erfindung zeigen.

[0051] Um die zahlreichen Kopien von Texturen zu vermeiden, zusammen mit der Zeit, die benötigt wird, um derartige Kopien zu erstellen, ebenso wie die große Menge von Speicherbetriebsmitteln, die derartige Kopien besetzen, liefert die vorliegende Erfindung eine Einrichtung zum Plazieren von Texturen aus dem Benutzerpuffer direkt in die Graphikhardware. Die Verwendung der vorliegenden Erfindung verbessert die Textur-Herunterladungsgeschwindigkeit ebenso wie das Aufbereitungsverhalten von Anwendungen, die sich dynamisch ändernde Texturen aufweisen. Dieselbe ermöglicht, daß Benutzer einer Texturabbildung ihre Texturen effizient direkt in den Hardware-Texel-Cache herunterladen.

[0052] Die Erfindung verwendet einen Mechanismus, um Texturen, die nur hardwaremäßig gespeichert sind, zu verfolgen, bewahrt jedoch die Fähigkeit, eine softwaremäßige Kopie der Textur zu liefern, wenn eine solche benötigt wird. Wenn folglich ein Benutzer die Textur wieder abfragt, oder wenn sich der Graphikkontext ändert, kann die Textur dem Benutzer wieder geliefert werden.

[0053] Wie oben erläutert wurde, speichern gegenwärtige Texturabbildungsimplementierungen eine Kopie der Benutzertextur softwaremäßig, um einen Mechanismus für eine Texturabfrage zu liefern, und um eine hardwaremäßige Zwischenspeicherung von Texeln zu ermöglichen, wenn nicht genügend Speicher existiert, daß alle Texel gleichzeitig in die Graphikhardware passen. In vielen Fällen wird diese Softwarekopie der Textur niemals benötigt. In diesen Fällen fragt der Benutzer die Textur nicht wieder ab, wobei während der Lebensdauer der Textur stets Raum existiert, um dieselbe in dem Texel-Cache zu speichern. Die vorliegende Erfindung optimiert diesen Fall, indem die Softwarekopie der Textur verzögert wird, bis dieselbe absolut benötigt wird, oder bis die Textur durch den Benutzer ausrangiert wird.

[0054] Zusätzlich verallgemeinert die vorliegende Erfindung das Texturherunterladen, um stets diesen Mechanismus zu verwenden, was die Softwarearchitektur stark vereinfacht und das Verhalten in Fällen wie z.B. Rahmenpufferkopien in den Textur-Cache erhöht – wobei wiederum keine Softwarekopie benötigt wird, es sei denn, eine solche wird beantragt.

[0055] Wie nun in dem Flußdiagramm von Fig. 5 gezeigt ist, spezifiziert gemäß dem vorliegenden erfinderischen Verfahren **50** das Anwendungsprogramm unter Verwendung eines API-Aufrufs **52** eine Textur, um ein mit einer Textur versehenes Dreieck zu zeichnen. Die API-Bibliothek verifiziert, daß der API-Aufruf gültig ist **54**, woraufhin dieselbe unmittelbar die Textur in die Graphikhardware herunterlädt **56**. Gemäß der vorliegenden Erfindung ist die Graphikhardwarekopie der Textur die einzige Kopie, die von der Textur gemacht wird, es sei denn, außergewöhnliche Dinge geschehen.

[0056] Gemäß der Erfindung wird eine Buchführung durchgeführt, um sowohl die Bibliothek als auch das Texturhintergrundprogramm zu informieren, daß sich die einzige Kopie der Textur in der Hardware **51** befindet. Diese Softwarekopie wird dann als "dirty" betrachtet, da sich die Textur nur in der Hardware befindet.

[0057] Das Anwendungsprogramm zeichnet dann ein Dreieck unter Verwendung der Textur, indem die Graphikhardware dasselbe einfach zeichnet **60**, da die Graphikhardware bereits eine gegenwärtige Kopie der Textur besitzt.

[0058] In dem Fall, in dem eine Textur aus der Graphikhardware wiedergewonnen werden muß, wurde gemäß dem alten Algorithmus zum Wiedergewinnen der Textur aus der Graphikhardware (entweder da der Benutzer ihren Wert anfordert, oder da die Graphikhardware eine andere Textur benötigt, die nicht paßt), die Textur entweder zu dem Benutzer zurückkopiert (Abfrage) oder dieselbe wurde in der Graphikhardware einfach überschrieben (neue Textur benötigt).

[0059] Wie nun in Fig. 6 gezeigt ist, überprüft das Anwendungsprogramm gemäß der vorliegenden Erfindung andererseits zuerst den Buchführungseintrag, um zu sehen, ob die Textur "dirty" ist **74**, d.h., ob dieselbe bereits in der Graphikhardware ist, wenn das Anwendungsprogramm eine Textur benötigt. Wenn dies der Fall ist, lädt das Texturhintergrundprogramm die Textur von der Hardware herauf **76**. Wenn die API-Bibliothek die Textur benötigt, gibt das Texturhintergrundprogramm dieselbe danach zu der API-Bibliothek **78**.

[0060] Für Fachleute auf dem Gebiet der Computergraphiken ist es offensichtlich, daß die Verwendung des Verfahrens gemäß der vorliegenden Erfindung

das Verhalten für Benutzer von Texturabbildungen stark verbessert, speziell für Benutzer, die Anwendungen mit sich dynamisch ändernden Texturtabellen besitzen. Unter Verwendung dieser Technik sind die Zeit und der Speicher, die benötigt werden, um eine Softwarekopie der Texturtabelle zu erstellen, beseitigt. Schlimmstenfalls sind dieselben beseitigt, bis der Benutzer den Texel-Cache-Raum überschreitet, ohne die Textur auszurangieren, oder bis die Textur aufgrund einer Benutzerabfrage oder einer anderen Software-Anforderung zurückgelesen wird. Im besten Fall sind die Zeit und der Speicher, die benötigt werden, vollständig beseitigt.

Patentansprüche

1. Verfahren zum Bereitstellen von Texturtabellen für die Erzeugung einer Graphik durch ein Computergraphiksystem, wobei das Computergraphiksystem einen Host-Computer mit zugeordnetem Systemspeicher und einer Graphikhardwarevorrichtung (**150**) zum Aufbereiten von mit einer Textur versehenen Bildern umfasst, wobei die Graphikhardwarevorrichtung (**150**) einen lokalen Speicher (**155**) aufweist und mit dem Host-Computer in Verbindung ist, und wobei der Host-Computer konfiguriert ist, um ein Anwendungsprogramm (PR1, PR2, PR3), eine Graphik-Anwendungsprogrammierschnittstelle (Graphik-API) (API1, API2, API3) und ein Hintergrundprogramm (**160**) auszuführen, wobei das Verfahren folgende Schritte aufweist:
(a) ansprechend auf einen gültigen Aufruf (**52**, **54**) der Graphik-API (API1, API2, API3) durch das Anwendungsprogramm (PR1, PR2, PR3), Laden (**56**) von durch das Anwendungsprogramm (PR1, PR2, PR3) angegebenen Texturtabellen in den lokalen Speicher (**155**) der Graphikhardwarevorrichtung (**150**) ohne eine Kopie der Texturtabellen in dem Systemspeicher des Host-Computer zu erzeugen;
(b) Setzen (**58**) einer Flag, um die Existenz und die Gültigkeit der in den lokalen Speicher (**155**) der Graphikhardwarevorrichtung (**150**) geladenen Texturtabellen anzuzeigen; und
(c) falls eine Kopie der Texturtabellen in dem Systemspeicher benötigt wird, Überprüfen (**72**, **74**) durch das Anwendungsprogramm (PR1, PR2, PR3), ob für die benötigten Texturtabellen die Flag gesetzt ist, und Kopieren (**76**) der Texturtabellen durch das Hintergrundprogramm (**160**) von dem lokalen Speicher (**155**) der Graphikhardwarevorrichtung (**150**) in den Systemspeicher des Host-Computers, falls die Flag gesetzt ist.

2. Verfahren gemäß Anspruch 1, bei dem der Schritt (c) das Weitergeben (**78**) der Texturtabellen an die Graphik-API (API1, API2, API3) umfasst.

3. Verfahren gemäß Anspruch 2, bei dem der Schritt des Setzens der Flag das Setzen einer Variable auf einen Wert aufweist, der anzeigt, ob sich die in dem lokalen Speicher (**155**) der Graphikhardwarevorrichtung (**150**) gehaltenen Texturtabellen geändert haben, seit eine Kopie derselben in der Graphik-API (API1, QAPI2, API3) zuletzt modifiziert wurde.

4. Verfahren Anspruch 3, das ferner den Schritt des Wiedererlangens der Texturtabellen aus dem lokalen Speicher (**155**) der Graphikhardwarevorrichtung (**150**) aufweist, wenn die Variable anzeigt, dass die Kopie geändert wurde.

Es folgen 6 Blatt Zeichnungen

Anhängende Zeichnungen

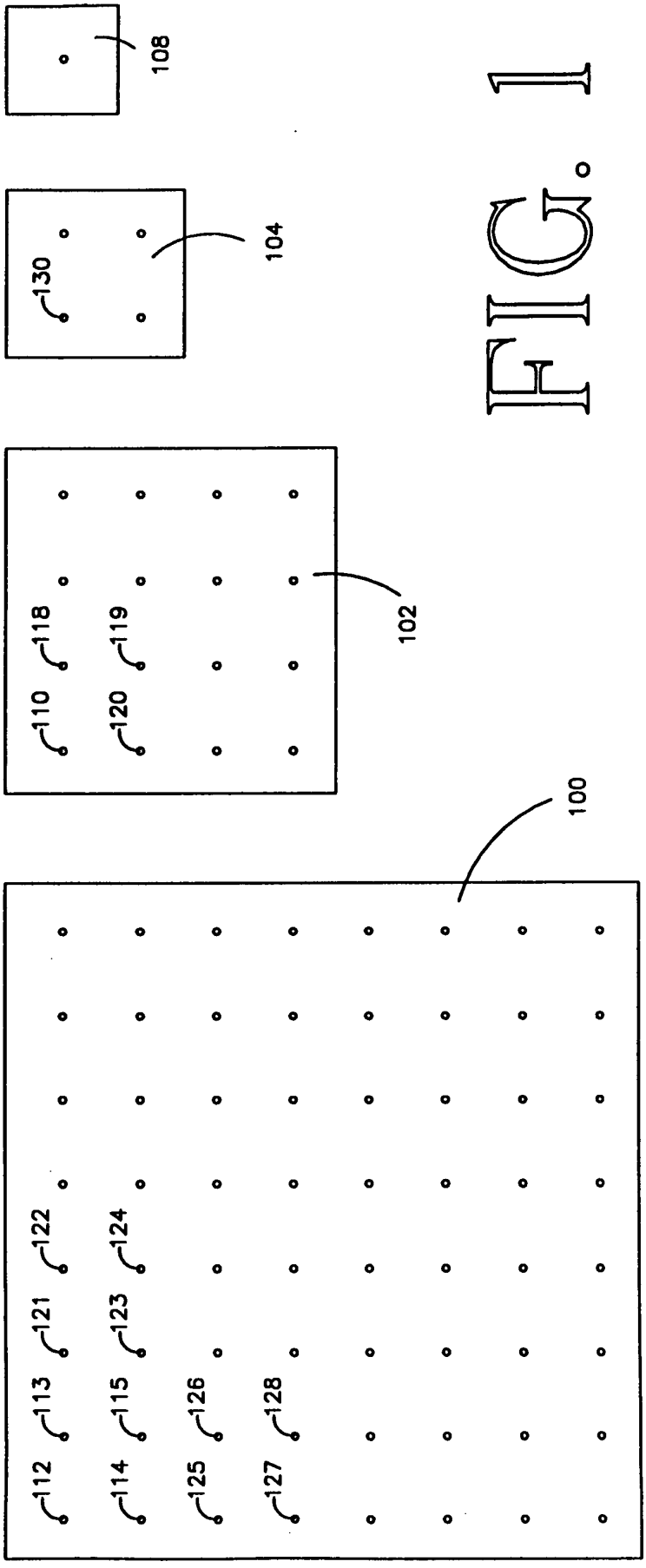


FIG. 1

FIG. 2 (STAND DER TECHNIK)

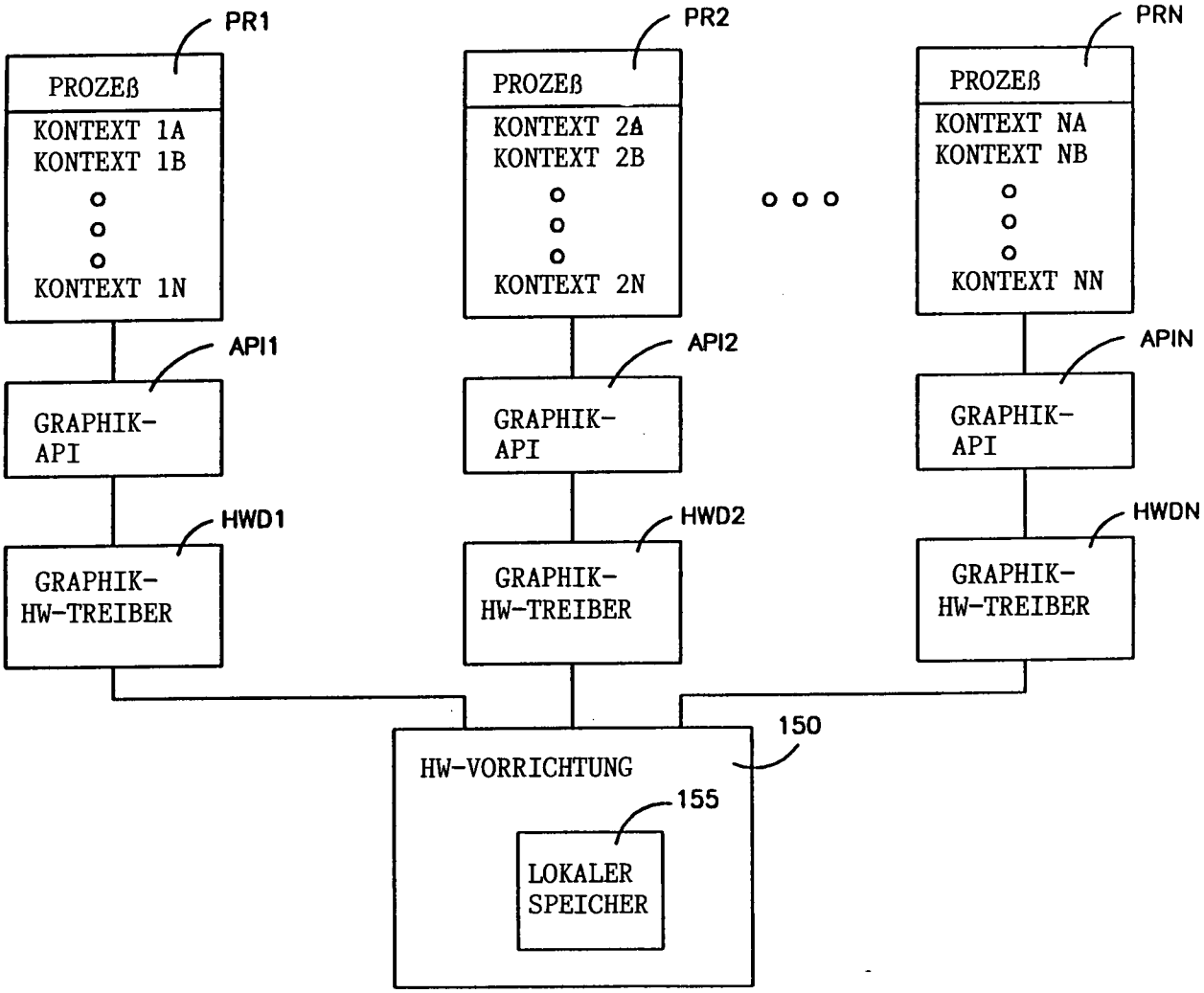


FIG. 3

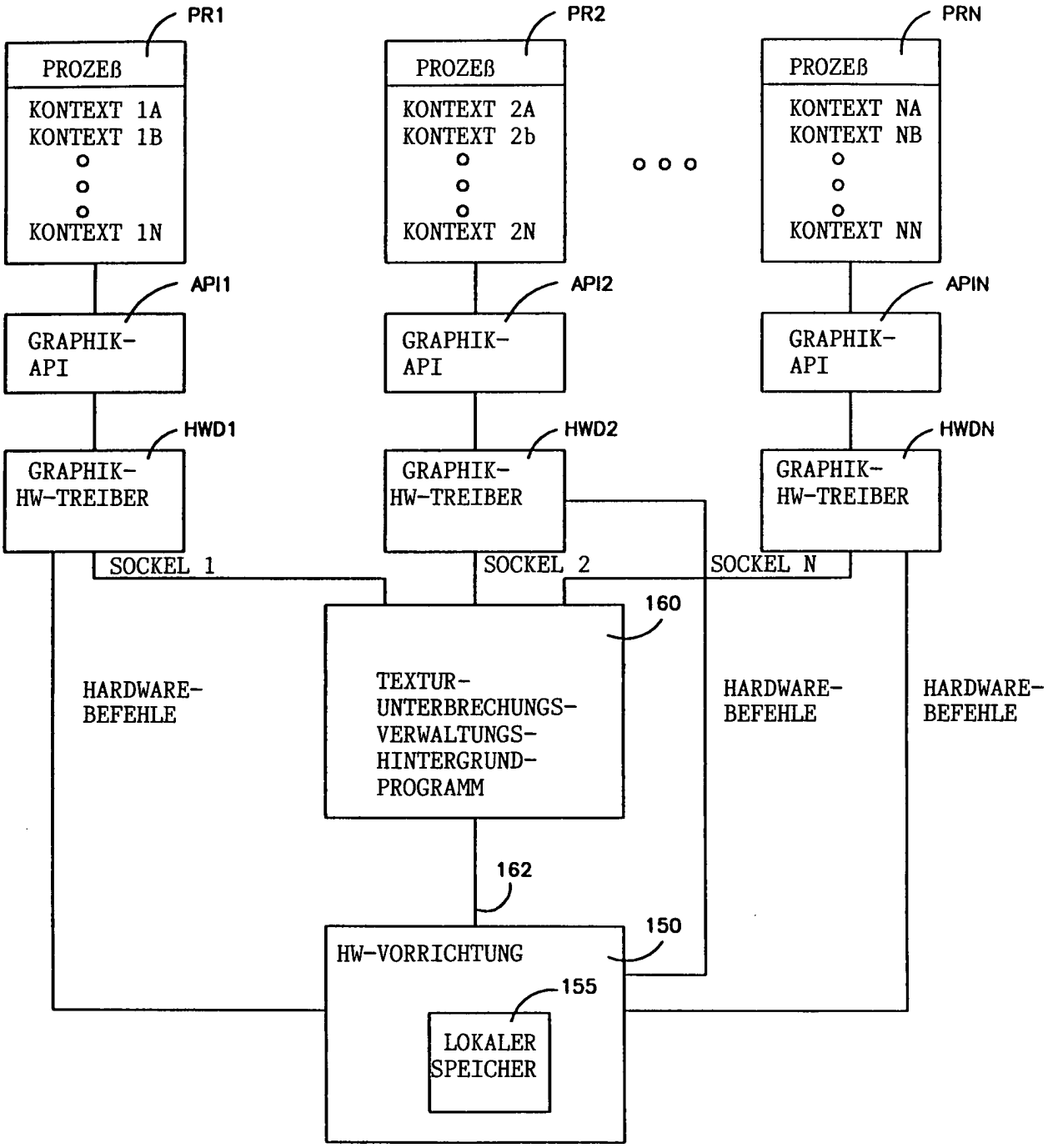

PR1
PROZEß
KONTEXT 1A
KONTEXT 1B
KONTEXT 1N
PR2
PROZEß
KONTEXT 2A
KONTEXT 2b
KONTEXT 2N
PRN
PROZEß
KONTEXT NA
KONTEXT NB
KONTEXT NN
API1
GRAPHIK-API
API2
GRAPHIK-API
APIN
GRAPHIK-API
HWD1
GRAPHIK-HW-TREIBER
HWD2
GRAPHIK-HW-TREIBER
HWDN
GRAPHIK-HW-TREIBER
SOCKEL 1
SOCKEL 2
SOCKEL N
160
TEXTUR-UNTERBRECHUNGS-VERWALTUNGS-HINTERGRUND-PROGRAMM
HARDWARE-BEFEHLE
HARDWARE-BEFEHLE
HARDWARE-BEFEHLE
162
150
HW-VORRICHTUNG
155
LOKALER SPEICHER

FIG. 4 (STAND DER TECHNIK)

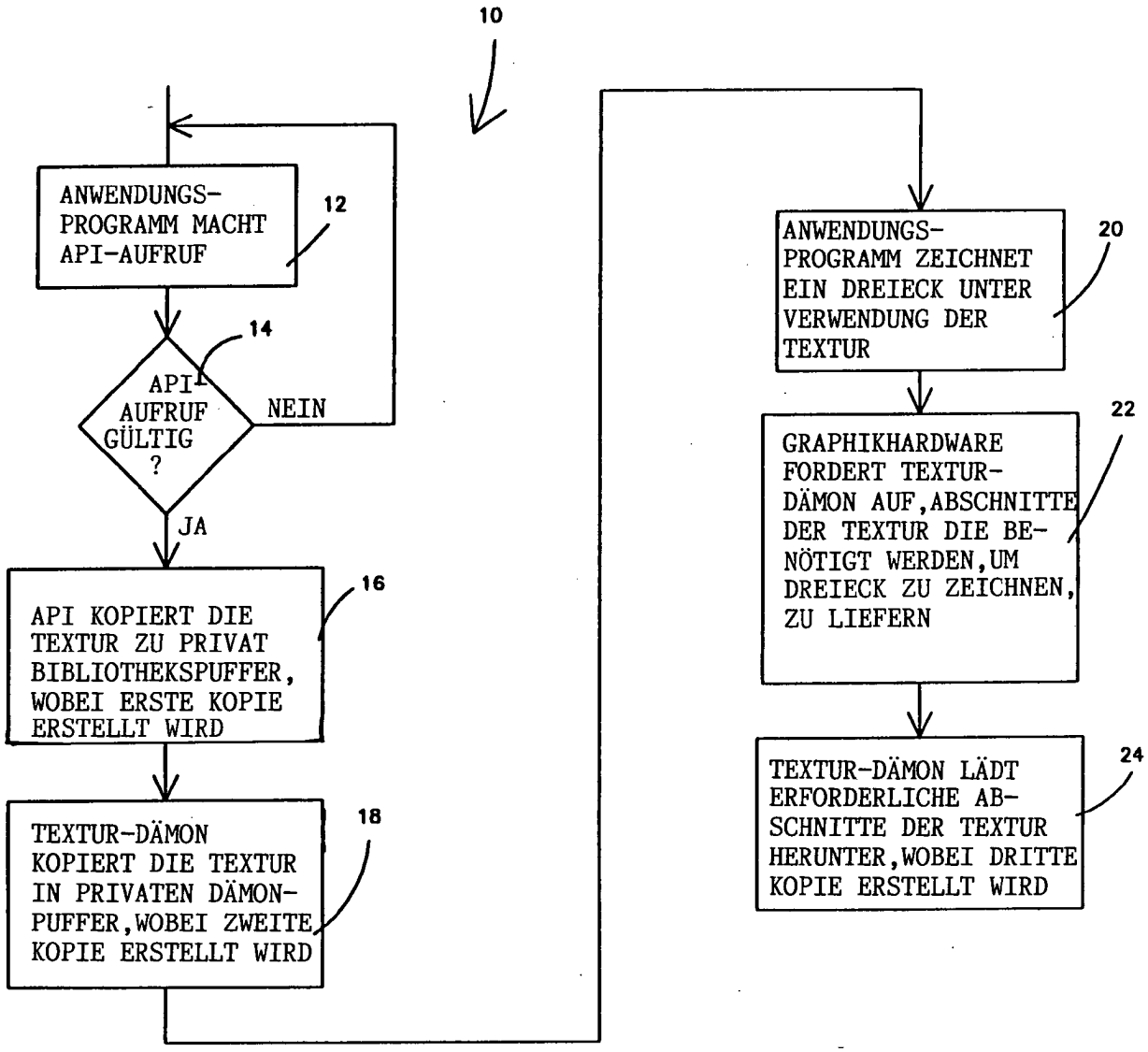


FIG. 5

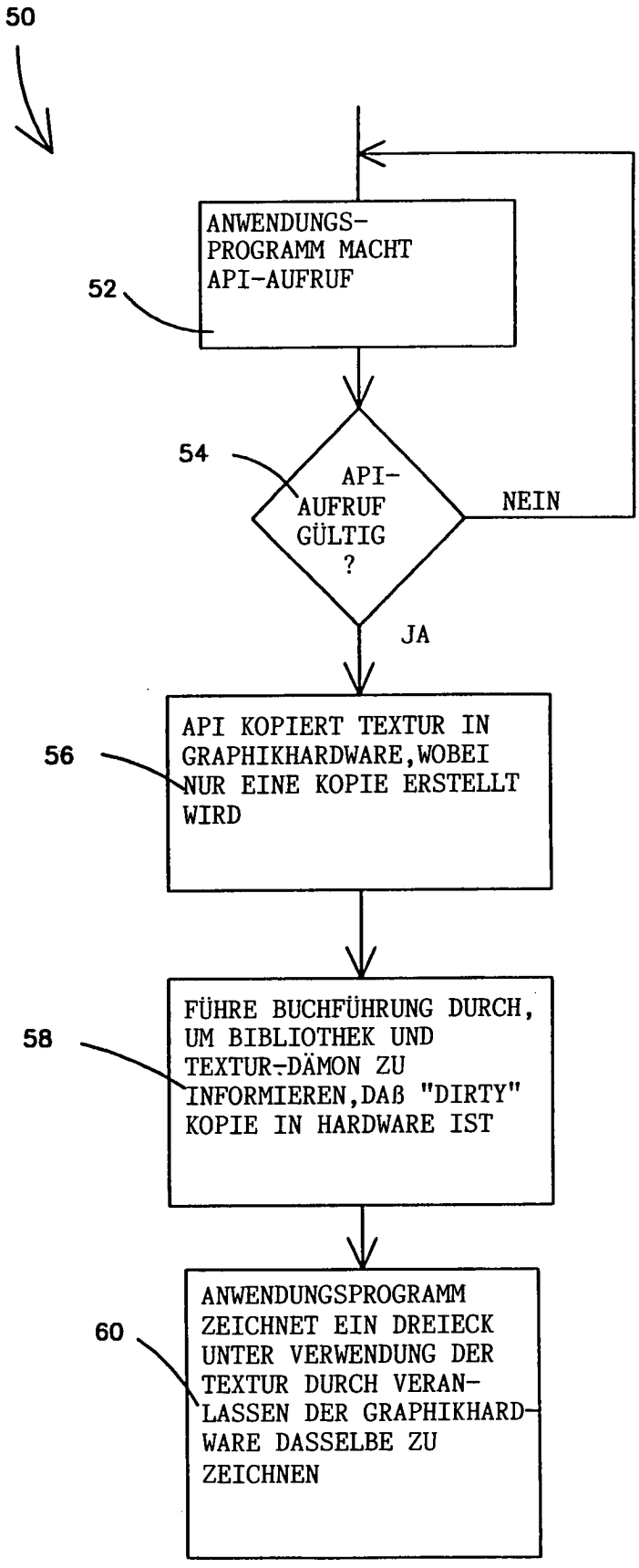
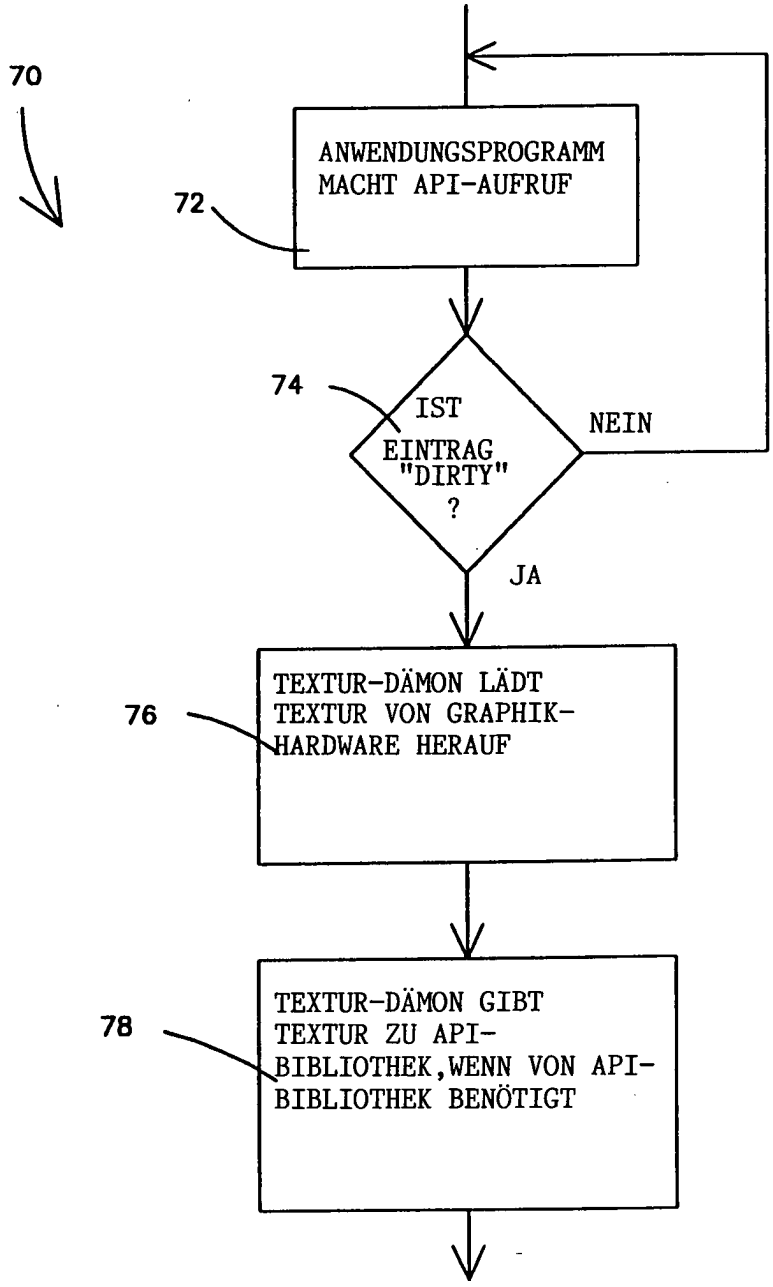

50
ANWENDUNGS-
PROGRAMM MACHT
API-AUFRUF
52
54
API-
AUFRUF
GÜLTIG
?
NEIN
JA
API KOPIERT TEXTUR IN
GRAPHIKHARDWARE,WOBEI
NUR EINE KOPIE ERSTELLT
WIRD
56
FÜHRE BUCHFÜHRUNG DURCH,
UM BIBLIOTHEK UND
TEXTUR-DÄMON ZU
INFORMIEREN,DAß "DIRTY"
KOPIE IN HARDWARE IST
58
ANWENDUNGSPROGRAMM
ZEICHNET EIN DREIECK
UNTER VERWENDUNG DER
TEXTUR DURCH VERAN-
LASSEN DER GRAPHIKHARD-
WARE DASSELBE ZU
ZEICHNEN
60

FIG. 6


70
ANWENDUNGSPROGRAMM MACHT API-AUFRUF
72
74
IST EINTRAG "DIRTY" ?
NEIN
JA
TEXTUR-DÄMON LÄDT TEXTUR VON GRAPHIK-HARDWARE HERAUF
76
TEXTUR-DÄMON GIBT TEXTUR ZU API-BIBLIOTHEK,WENN VON API-BIBLIOTHEK BENÖTIGT
78