(19) Bundesrepublik Deutschland
Deutsches Patent- und Markenamt

(10) **DE 197 23 063 B4** 2009.05.07

(12) **Patentschrift**

(21) Aktenzeichen: **197 23 063.6**
(22) Anmeldetag: **02.06.1997**
(43) Offenlegungstag: **29.01.1998**
(45) Veröffentlichungstag der Patenterteilung: **07.05.2009**

(51) Int Cl.[8]: ***G06T 1/60*** (2006.01)
G06F 12/08 (2006.01)

(30) Unionspriorität:
08/690,432 26.07.1996 US

(62) Teilung in:
197 58 921.9

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

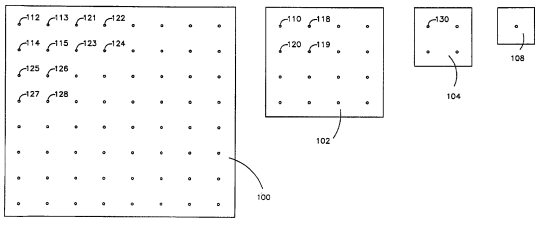
(74) Vertreter:
Schoppe, F., Dipl.-Ing.Univ., Pat.-Anw., 82049 Pullach

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

(56) Für die Beurteilung der Patentfähigkeit in Betracht gezogene Druckschriften:
EP 06 68 555 A2
EP 04 47 227 A2
PRESS, W.H.: Numerical Recipes in C - The Art of S
cientific Computing, Cambridge University Press, 1
992, S. 338-341; Parallel in Place Graphics Buffer Reorganizations, IBM Technical Disclosure Bulleti n, Vol. 37, No. 7, July 1994, S. 251-258; SEGAL M. , AKELEY K.: The OpenGL TM Graphics System: A Spec
ification, (Version 1.0), 1993, Kapitel 3.8, Textu ring, S. 78-89; IEEE-CS TC-RTS Newsletter for Sun,
Mar 26, 1995, S. 1, 33-36; MOLNAR, S.: >>The Pixe lFlow Texture and Image Subsystem<<, Proceedings o
f the 10th Eurographics Workshop on Graphics Hardw
are, Maastricht, The Netherlands, Aug. 28-29, 1995 , pp. 3-13;

(54) Bezeichnung: **Verfahren zum Speichern von Texeldaten einer Textur in einem zusammenhängenden Speicherblock**

(57) Hauptanspruch: Verfahren zum Speichern von Texeldaten einer Textur in einem zusammenhängenden Speicherblock eines Speichers einer Texturabbildungshardware in einem Computergraphiksystem, wobei einer Textur eine MIP-Abbildung mit einer Mehrzahl von MIP-Abbildungsebenen (100, 102, 104, 108) zugeordnet ist, wobei das Verfahren folgende Schritte aufweist
(a) Empfangen (12) von Texeldaten für eine erste heruntergeladene MIP-Abbildungsebene (100, 102, 104, 108);
(b) Berechnen (16) eines zusammenhängenden Speicherblocks für eine vollständige MIP-Abbildung, basierend auf der Größe und der Ebenennummer der ersten heruntergeladenen MIP-Abbildungsebene (100, 102, 104, 108);
(c) Zuordnen (18) des zusammenhängenden Speicherbocks für die vollständige MIP-Abbildung;
(d) Bestimmen (33) eines Versatzwertes in dem zusammenhängenden Speicherblock, basierend auf der Ebenennummer der ersten heruntergeladenen MIP-Abbildungsebene (100, 102, 104, 108), wobei der Versatzwert der vorbestimmten Position zum Speichern von Texeldaten entspricht, die der ersten heruntergeladenen MIP-Abbildungsebene (100, 102, 104, 108) zugeordnet sind;
(e) Speichern (39) der Texeldaten für die erste heruntergeladene MIP-Abbildungsebene (100, 102,...



Beschreibung

[0001] Die vorliegende Erfindung bezieht sich auf ein Verfahren zum Speichern von Texeldaten einer Textur in einem zusammenhängenden Speicherblock. Insbesondere bezieht sich die Erfindung auf eine Softwarespeicherverwaltung von Texturabbildungen eines Texturabbildungs-Computergraphiksystems und insbesondere auf einen neuen Lösungsansatz, welcher das System beträchtlich beschleunigt, indem sichergestellt wird, daß die gesamten Texturdaten in einem zusammenhängenden Speicher gehalten werden.

[0002] Gegenwärtige Implementationen der Texturabbildung, wie sie detaillierter in der Europäischen Patentanmeldung EP 0 749 100 A beschrieben ist und deren Inhalt hierin durch Bezugnahme aufgenommen ist, speichern eine Kopie der Textur des Benutzers in Software, um eine Vorrichtung zur Texturabfrage zu schaffen, und um eine Cache-Speicherung von Texeln oder Texturelementen in Hardware zu ermöglichen, wenn nicht genug Speicher vorhanden ist, damit alle Texel zu einem Zeitpunkt in die Hardware passen.

[0003] Bei typischen Computergraphiksystemen wird ein Objekt, das auf dem Anzeigebildschirm dargestellt werden soll, in eine Mehrzahl von Graphikgrundelementen gebrochen. Grundelemente sind Grundkomponenten eines Graphikbildes und können Punkte, Linien, Vektoren und Polygone, wie z. B. Dreiecke, umfassen. Typischerweise ist ein Hardware/Software-Schema implementiert, um auf dem zweidimensionalen Anzeigebildschirm die Graphikgrundelemente, die die Ansicht eines oder mehrerer Objekte darstellen, die auf dem Bildschirm dargestellt werden, aufzubereiten oder zu zeichnen.

[0004] Typischerweise werden die Grundelemente, die das dreidimensionale Objekt definieren, das aufbereitet werden soll, von einem Hostcomputer geliefert, welcher jedes Grundelement in Grundelement-Daten definiert. Wenn das Grundelement beispielsweise ein Dreieck ist, kann der Hostcomputer das Grundelement in den Koordinaten x, y, z seiner Spitzen sowie in den R-, G-, B-Farbwerten jeder Spitze definieren. Eine Aufbereitungshardware interpoliert die Grundelementdaten, um die Anzeigebildschirmpixel zu berechnen, die eingeschaltet werden, um jedes Grundelement darzustellen, und um die R-, G-, B-Werte für jedes Pixel zu berechnen.

[0005] Frühere Graphiksysteme schafften es nicht, Bilder auf eine ausreichend realistische Art und Weise anzuzeigen, um komplexe dreidimensionale Objekte darzustellen oder zu modellieren. Die Bilder, die von solchen Systemen angezeigt werden, zeigten außerordentlich weiche Oberflächen ohne Texturen, Stöße, Kratzer, Schatten oder andere Oberflächendetails, die in dem modellierten Objekt vorhanden sind.

[0006] Als Ergebnis wurden Verfahren entwickelt, um Bilder mit verbesserten Oberflächendetails 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 anschließend das Abbilden des texturierten dreidimensionalen Objekts auf den zweidimensionalen Graphikanzeigebildschirm betrifft, um das resultierende Bild anzuzeigen. Oberflächendetailattribute, welche üblicherweise Textur-abgebildet sind, umfassen eine Farbe, eine spieglige Reflexion, eine Vektorstörung, eine Spiegligkeit, eine Transparenz, Schatten, Oberflächenungleichmäßigkeiten und Abstufungen.

[0007] Die Texturabbildung betrifft das Anlegen von einem oder mehreren Punkttexturelementen ("Texeln") an jedes Punktelement ("Pixel") des angezeigten Abschnitts des Objekts, zu dem die Textur abgebildet wird. Die Texturabbildungshardware ist üblicherweise mit Informationen versehen, die die Art und Weise anzeigen, auf die die Texel in einer Texturabbildung den Pixeln auf dem Anzeigebildschirm, die das Objekt darstellen, entsprechen. Jedes Texel in einer Texturabbildung wird durch Koordinaten S und T definiert, welche seine Position in der zweidimensionalen Texturabbildung identifizieren. Für jedes Pixel wird auf das entsprechende Texel oder auf die entsprechenden Texel, die auf dasselbe abgebildet werden, von der Texturabbildung zugegriffen, und dieselben werden in die endgültigen R-, G-, B-Werte aufgenommen, die für das Pixel erzeugt werden, um das texturierte Objekt auf dem Anzeigebildschirm darzustellen.

[0008] Es sollte offensichtlich sein, daß jedes Pixel in einem Objektgrundelement nicht in einer Eins-zu-Eins-Korrespondenz mit einem einzelnen Texel in der Texturabbildung für jede Ansicht des Objekts abgebildet werden kann. Je näher das Objekt beispielsweise an dem Anzeigetor, das auf dem Anzeigebildschirm dargestellt ist, positioniert ist, um so größer wird das Objekt erscheinen. Sowie das Objekt auf dem Anzeigebildschirm größer erscheint, wird die Darstellung der Textur detaillierter. Wenn das Objekt somit einen ziemlich großen Abschnitt des Anzeigebildschirms einnimmt, wird eine große Anzahl von Pixeln verwendet, um das Objekt auf dem Anzeigebildschirm darzustellen, wobei jedes Pixel, das dem Objekt entspricht, in eine Eins-zu-Eins-Korrespondenz mit einem einzigen Texel in der Texturabbildung abgebildet werden kann, oder

ein einzelnes Texel kann auf viele Pixel abgebildet werden. Wenn das Objekt jedoch einen relativ kleinen Abschnitt des Anzeigebildschirms einnimmt, wird eine viel kleinere Anzahl von Pixeln verwendet, um das Objekt darzustellen, was darin resultiert, daß die Textur weniger detailliert dargestellt ist, derart, daß jedes Pixel in mehrere Texel abgebildet werden kann. Zusätzlich kann jedes Pixel in mehrere Texel abgebildet werden, wenn eine Textur auf einen kleinen Abschnitt eines Objekts abgebildet wird. Resultierende Texeldaten werden für jedes Pixel berechnet, das auf mehr als ein Texel abgebildet wird. Da es üblich ist, daß ein Pixel auf mehrere Texel abgebildet wird, stellen resultierende Texeldaten für ein Pixel typischerweise einen Mittelwert der Texel, die auf das Pixel abgebildet werden, dar.

[0009] Texturabbildungshardwaresysteme umfassen typischerweise einen lokalen Speicher, der Daten speichert, die einer Textur entsprechen, die mit dem aufzubereitenden Objekt verknüpft ist. Wie es oben erörtert wurde, kann ein Pixel zu mehreren Texeln abgebildet werden. Wenn es nötig wäre, daß die Texturabbildungshardware eine große Anzahl von Texeln liest, die auf ein Pixel von dem lokalen Speicher abgebildet werden, um einen Durchschnittswert zu erzeugen, würde eine große Anzahl von Speicherlesevorgängen und das Bilden des Durchschnitts von vielen Texelwerten erforderlich sein, was zeitaufwendig sein würde und das Systemverhalten verschlechtern würde.

[0010] Um dieses Problem zu überwinden, wurde ein Schema entwickelt, das das Erzeugen einer Serie von Abbildungen, die "MIP"-Abbildungen genannt werden (MIP bedeutet "Multum In Parvo" = viele Dinge in einem kleinen Platz), für jede Textur betrifft, wobei die MIP-Abbildungen der Textur, die mit dem aufzubereitenden Objekt verknüpft ist, in dem lokalen Speicher der Texturabbildungshardware gespeichert werden. Eine MIP-Abbildung für eine Textur umfaßt eine Basis-("Ebene-0"-)Abbildung, die direkt der Texturabbildung entspricht, sowie eine Serie von gefilterten Abbildungen, wobei jede aufeinanderfolgende Abbildung größenmäßig um einen Faktor 2 in jeder der zwei Texturabbildungsdimensionen reduziert ist. Ein veranschaulichendes Beispiel eines Satzes von MIP-Abbildungen ist in Fig. 1 gezeigt. Die MIP-Abbildungen umfassen eine Basisabbildung ("Ebene 0") **100**, die eine Größe von 8 × 8 Texel aufweist, sowie eine Serie von Abbildungen **102**, **104** und **108**, die die Ebene 1, welche 4 × 4 Texel ist, die Ebene 2, welche 2 × 2 Texel ist bzw. die Ebene 3 darstellen, welche eine Größe von einem Texel aufweist.

[0011] Die Ebene-1-Abbildung **102**, die eine Größe von 4 × 4 aufweist, wird erzeugt, indem die Basisabbildung **100** Kasten-gefiltert (dezimiert) wird, derart, daß jedes Texel in der Ebene-1-Abbildung **102** einem Durchschnitt von vier Texeln von der Ebene-0-Basisabbildung **100** entspricht. Das Texel **110** in der Ebene-1-Abbildung **102** gleicht beispielsweise dem Durchschnitt der Texel **112** bis **115** in der Ebene-0-(Basis-)Abbildung **100**. Auf ähnliche Weise gleichen die Texel **118** und **120** in der Ebene-1-Abbildung **102** den Durchschnitten der Texel **121** bis **124** bzw. **125** bis **128** in der Ebene-0-(Basis-)Abbildung **100**. Die 2 × 2-Abbildung **104** (die Ebene-2-Abbildung) wird durch Kastenfiltern der Ebene-1-Abbildung **102** erzeugt, derart, daß ein Texel **130** in der Ebene-2-Abbildung **104** dem Durchschnitt der Texel **110** und **118** bis **120** in der Ebene-1- Abbildung **102** gleicht. Das einzige Texel in der Ebene-3-Abbildung **108** wird erzeugt, indem die vier Texel in der Ebene-2-Abbildung **104** gemittelt werden.

[0012] Herkömmliche Graphiksysteme laden allgemein von dem Hauptspeicher des Hostcomputers die vollständige Serie von MIP-Abbildungen für jede Textur, die mit den auf dem Anzeigebildschirm aufzubereitenden Grundelementen verwendet werden soll, in den lokalen Speicher der Texturabbildungshardware. Wie es für Fachleute verständlich ist, meint eine komplette Serie von MIP-Abbildungen alle MIP-Abbildungen von der Ebene 0 bis zur Ebene N, wobei die Ebene N eine 1 × 1-MIP-Abbildung ist. Somit kann die Texturabbildungshardware auf Texturdaten von irgendeiner der Ebenen der Serie von MIP-Abbildungen zugreifen. Die Bestimmung, auf welche Abbildung zugegriffen wird, um die Texeldaten für ein spezielles Pixel zu liefern, basiert auf der Anzahl von Texeln, auf die das Pixel abgebildet wird. Wenn das Pixel beispielsweise in einer Eins-zu-Eins-Korrespondenz mit einem einzigen Texel in der Texturabbildung abgebildet wird, dann wird auf die Basisabbildung **100** zugegriffen. Wenn das Pixel jedoch auf 4, 16 oder 64 Texel abgebildet wird, dann wird auf die Abbildungen **102**, **104** bzw. **108** zugegriffen, da diese Abbildungen jeweils Texeldaten speichern, die einen Durchschnitt von 4, 16 und 64 Texeln in der Texturabbildung darstellen.

[0013] Wie es erkannt werden wird, kann eine Serie von Textur-MIP-Abbildungen eine große Menge an Systemsoftwarespeicher zur Speicherung erfordern. Eine Serie von MIP-Abbildungen für eine Textur mit einer Texturbasisabbildung von 1.024 × 1.024 Texeln erfordert mehr als fünf Megabyte an Systemsoftwarespeicher, um eine Kopie der MIP-abgebildeten Textur zu speichern. Somit verwenden die mehreren gespeicherten Kopien der MIP-abgebildeten Textur eine wesentliche Menge an Systemsoftwarespeicher.

[0014] Während der Systemsoftwarespeicher in der Lage sein kann, bis zu ein paar Gigabytes Softwaredaten

zu speichern, besteht ein anderer Punkt, welcher angegangen werden muss, darin, wo die MIP-Abbildungen tatsächlich gespeichert werden. Um eine Aufbereitung von Graphikbildern mit hoher Geschwindigkeit zu erreichen, ist es insbesondere wichtig, in der Lage zu sein, die Texelinformationen von der geeigneten Ebene der MIP-Abbildung zu der Graphikanzeige so schnell als möglich zu übertragen. Obwohl es am besten wäre, wenn die Position aller Ebenen von vornherein bekannt sein würde, macht die Art und Weise, auf die eine typische Graphikanwendungsprogrammierschnittstelle ("API"; API = Application Programmer Interface) arbeitet, dieses zu einer speziell schwierigen Aufgabe. Insbesondere erlaubt eine Graphik-API, die "OpenGL" genannt wird und von Hewlett Packard erhältlich ist, daß der Benutzer MIP-Abbildungsebenen derart herunterlädt, daß die verschiedenen Ebenen in einer beliebigen (Ebenen-)Reihenfolge zu dem Speicher gesendet werden können. Siehe z. B. Segal M., Akeley K.: „The OpenGL™ Graphics System: A Specification (Version 1.0)", 1993, Kapitel 3.8, Texturing, Seiten 78 bis 89.

[0015] Bisher wurden solche MIP-Abbildungsebenen einzeln im Speicher gespeichert, wobei als Position für jede der Ebenen der Speicher verwendet wird, der von einer Betriebssystemspeicherzuordnungsroutine ("Mallocing"-Routine; Mallocing = Memory ALLOCatING) zurückgegeben wird. Somit wurden die tatsächlichen Positionen in Speicher, in denen Ebenen der MIP-Abbildung gespeichert werden, dem Betriebssystem überlassen. Wenn daher eine einzelnen Ebene benötigt wurde, musste sie geortet werden, was in einer allgemeinen Verlangsamung des Betriebs des Systems resultierte.

[0016] Die EP 0 447 227 A beschreibt ein Verfahren und eine Vorrichtung zur Erzeugen von texturierten Graphikgrundelementen in einem Computergraphiksystem mit einem Rahmenpuffer. Zunächst wird für eine Oberfläche eine zweidimensionale ursprüngliche Texturabbildung bestimmt und in dem Rahmenpuffer abgespeichert. Anschließend wird die ursprüngliche Texturabbildung unabhängig voneinander in zwei Richtungen neu abgetastet, unter Verwendung eines asymmetrischen Filters, um vielfache Versionen einer Textur zu schaffen, und um texturierte Pixel auf einer Anzeige in dem Rahmenpuffer zu adressieren. Die texturierten Pixel werden Bereichen in dem Rahmenpuffer zugeordnet und die texturierten Graphikgrundelemente werden angezeigt.

[0017] Die Aufgabe der vorliegenden Erfindung besteht darin, ein Verfahren zum Speichern von Texturdaten zu schaffen, das es ermöglicht, Texturdaten einer MIP-Abbildung unabhängig von der Reihenfolge der Bereitstellung der Texeldaten für unterschiedliche MIP-Abbildungsebenen und unabhängig davon, ob Texeldaten für andere Texturen bereitgestellt werden, für eine schnelle und zuverlässige Wiedergewinnung zu speichern.

[0018] Diese Aufgabe wird durch ein Verfahren gemäß Anspruch 1 gelöst.

[0019] Gemäß dem bevorzugten Ausführungsbeispiel der Erfindung wird eine Vorrichtung geschaffen, mit der es möglich ist, Ebenen einer MIP-verarbeiteten OpenGL-Texturabbildung in einem zusammenhängenden Speicher zu speichern, während die Datenintegrität beibehalten wird. Somit reduziert oder eliminiert die vorliegende Erfindung Speichercachefehlschläge, wenn eine vollständig MIP-verarbeitete Texturabbildung zur Hardware heruntergeladen wird, oder wenn eine Texturabbildung eine Softwarerasterisierung verwendet. Ebenfalls hält die vorliegende Erfindung die Integrität der heruntergeladenen Daten bei, selbst wenn die Daten nicht in die Beschreibung der gegenwärtigen vollständigen MIP-Abbildung passen.

[0020] Gemäß der Erfindung wird ein Algorithmus geschaffen, welcher in der Lage ist, den Gesamtspeicher zu berechnen, der benötigt wird, um eine volle MIP-Abbildung zu speichern, und zwar basierend auf der ersten Ebene, die zu dem Graphikkern geleitet wird, sowie basierend auf folgenden Basisabbildungsebenenänderungen. Jede Ebene wird dann in dem zusammenhängenden Speicher gespeichert, wenn die Ebene gültig ist, oder in einer temporären Speicherposition, wenn die Ebene nicht gültig ist. Jedesmal, wenn sich die Basisebene verändert, werden alle Ebenen nach ihrer Gültigkeit getestet, wobei die gültigen Ebenen in dem zusammenhängenden Speicher plaziert werden.

[0021] Bevorzugte Ausführungsbeispiele der vorliegenden Erfindung werden nachfolgend bezugnehmend auf die beiliegenden Zeichnungen detaillierter erörtert. Es zeigen:

[0022] Fig. 1 eine Graphikdarstellung eines Satzes von Textur-MIP-Abbildungen; und

[0023] Fig. 2 bis Fig. 6 ein Flußdiagramm, das das Verfahren der vorliegenden Erfindung darstellt.

[0024] Dieser Beschreibung beigefügt und in die Beschreibung aufgenommen ist ein Anhang, welcher den Quellencode (in der Programmiersprache C) für das bevorzugte Ausführungsbeispiel der Erfindung enthält. Obwohl davon ausgegangen wird, daß der Quellencode die Details der Erfindung ausreichend beschreibt, der-

art, daß ein Fachmann auf dem Gebiet der Computergraphik in der Lage sein wird, die vorliegende Erfindung ohne weiteres zu verstehen, werden zusätzliche Details der Erfindung nachfolgend bezugnehmend auf die Flußdiagramme der Fig. 2 bis Fig. 6 beschrieben, welche das Verfahren darstellen, das von der Erfindung verwendet wird.

[0025] Bezugnehmend auf Fig. 2 wird das Flußdiagramm allgemein als das Gesamtverfahren **10** bezeichnet. Gemäß der vorliegenden Erfindung **10** sollen die Texturdaten beim Durchführen des erfindungsgemäßen Verfahrens in einen zusammenhängenden Block eines Speichers plaziert werden. Wie es oben dargelegt wurde, bestehen die Texturdaten aus den Daten, die mit allen Ebenen von der Ebene 0 bis zu Ebene n, wobei die Ebene n ein 1 × 1-Array ist, verknüpft sind. Zur Erklärung ist es ausreichend, zuerst festzustellen, daß die erste Sache, die bestimmt werden muß, darin besteht, ob ein ausreichender Speicher existiert, um alle Texeldaten in einen einzigen Speicherblock zu plazieren. Bezugnehmend auf Fig. 2 ist diese Bestimmung durch einen Entscheidungsblock **14** dargestellt. Wenn es nicht bekannt ist, ob ausreichend Speicher vorhanden ist, dann muß die Größe des zusammenhängenden Speicherblocks berechnet werden, welcher benötigt werden wird (**16**), und der Speicher muß zugeordnet werden (**18**). Folgendes Beispiel sei genannt: wenn die Ebene-0-MIP-Abbildung eine 8 × 8-Abbildung ist, besetzt dieselbe 64 "Positionen", wobei die tatsächliche Speichermenge durch die Anzahl von Bytes pro Texel (mal 64) bestimmt werden würde. Somit würde die Ebene-1-MIP-Abbildung 4 × 4 oder 16 Positionen besetzen, während die Ebene-2-MIP-Abbildung 2 × 2 oder vier Positionen besetzen würde, und die Ebene-3-MIP-Abbildung 1 × 1 oder eine Position besetzen würde. Somit würde die Gesamtanzahl von Positionen 85 "Positionen" mal der Anzahl von Bytes pro Texel sein. Wie es zu sehen ist, bedeutet die Kenntnis, daß die Ebene-2-MIP-Abbildung eine 2 × 2-Abbildung ist, daß die Basisabbildung eine 8 × 8-Abbildung ist. Wenn also die Ebene und die Größe einer MIP-Abbildung gegeben ist, ist es ohne weiteres möglich, die Größe eines zusammenhängenden Speicherblocks zu bestimmen, welcher benötigt werden wird, um die volle MIP-Abbildung zu speichern.

[0026] Wenn es bestimmt wird, daß kein ausreichender Speicher zugeordnet werden kann **20**, wird eine Fehlerbedingung **22** resultieren. Alternativ werden die Basis-Abbildungs-(Ebene-0-)Werte gespeichert werden **24**. Anschließend wird eine Flag, die "LevelOK" (LevelOK = Ebene in Ordnung) genannt wird, bei dem bevorzugten Ausführungsbeispiel der Erfindung bei **26** auf "0" eingestellt, wonach eine Überprüfung durchgeführt wird, um zu bestimmen, ob die Ebenen-Informationen in Ordnung sind **28**. Dies bedeutet, daß eine Bestimmung durchgeführt wird, ob die Informationen, die mit der MIP-Abbildungsebene, die geladen ist, verknüpft sind, mit den Informationen in Einklang sind, die vorher über die MIP-Abbildung bekannt waren. Wenn beispielsweise eine "Ebene-1"-MIP-Abbildung mit einer Größe von 4 × 4 heruntergeladen wurde, und wenn dann eine Ebene-0-MIP-Abbildung mit einer Größe von 8 × 8 heruntergeladen wird, würden die Daten in Einklang sein, wobei die Ebene in Ordnung sein würde. Wenn alternativ eine "Ebene-1"-MIP-Abbildung mit einer Größe von 4 × 4 heruntergeladen wird, und wenn dann eine Ebene-0-MIP-Abbildung mit einer Größe von 4 × 4 heruntergeladen wird, würden die Daten nicht in Einklang sein, und die Ebene würde nicht in Ordnung sein. Wenn es bestimmt wird, daß die Daten mit vorher heruntergeladenen Daten in Einklang sind, wird die LevelOK-Flag auf "1" gesetzt **30**.

[0027] Wenn die LevelOK-Flag "0" ist, wird eine Überprüfung durchgeführt **32**, um zu bestimmen, ob die MIP-Abbildung für die Basisabbildung (Ebene 0) war. Wenn dies nicht so ist, oder wenn es so ist und die LevelOK-Flag gleich "1" war, wird zu einem Punkt "C" **36** gegangen (siehe Fig. 5). Wenn die MIP-Abbildung für die Basisabbildung (Ebene 0) ist, und wenn die LevelOK-Flag auf "0" eingestellt ist, d. h. Schritt **34**, wird zu einem Punkt "A" **38** gegangen (siehe Fig. 3), wobei die Basisebeneninformationen **50** und ein Zeiger auf einen existierenden Speicher **40** gespeichert werden.

[0028] Weiter bezugnehmend auf Fig. 3 wird nun die Größe eines zusammenhängenden Speichers berechnet **42**, und der zusammenhängende Speicher wird zugeordnet **44**. Nach einem Test, um zu bestimmen, ob der zusammenhängende Speicher korrekt zugeordnet wurde **46**, ist nichts mehr zu tun, wenn es bestimmt wird, daß ein Fehler auftrat **48**. Ein Zähler **52**, eine Inkrementiereinrichtung **54** und eine Testprozedur **56** werden eingestellt, um eine wiederholte Iteration durch eine Schleife zu erlauben. Bei dem bevorzugten Ausführungsbeispiel der Erfindung umfasst die Schleife die Schritte des Erhaltens eines Zeigers auf die gegenwärtige Ebene **62** und des Bestimmens, daß die gegenwärtige Ebene noch nicht auf eine Position in dem Speicher **64** zeigt. Diese Schleife wird wiederholt iteriert, und der Schleifenzähler **54** wird jedesmal inkrementiert, bis der Schleifenzähler anzeigt, daß die Schleife die maximale Anzahl von möglichen unterstützten Ebenen iteriert worden ist. Bei dem gegenwärtigen Ausführungsbeispiel der Erfindung werden basierend auf einer gegenwärtig verwendeten Hardware und basierend auf Speicherkapazitäten nicht mehr als 16 Ebenen unterstützt. Demgemäß kann die Schleife nicht mehr als 15 mal verarbeitet werden, obwohl es für Fachleute offensichtlich sein wird, daß diese Anzahl vergrößert werden würde, wenn eine zukünftige Hardware- und Speicher-Verfügbarkeit

die Verwendung einer größeren Anzahl von MIP-Abbildungsebenen vorschreiben würden.

[0029] Wenn es in dem Entscheidungskasten **64** bestimmt wird, daß die gegenwärtige Ebene auf einen Speicher zeigt, dann wird die Größe der gegenwärtigen Ebene berechnet **70**, und es wird eine Variable, d. h. LevelOK, die anzeigt, daß die Ebenendaten korrekt sind, in einem Schritt **72** auf "0" initialisiert, was anzeigt, daß es nicht bekannt ist, ob die Daten korrekt sind.

[0030] Die Ebenendaten werden nun bezüglich ihrer Genauigkeit **74** getestet, und wenn eine Bestimmung durchgeführt wird, daß die Ebenendaten genau sind, wird in einem Schritt **88** LevelOK auf "1" eingestellt, was anzeigt, daß die Ebenendaten genau sind, und es wird ein Versatz in dem zusammenhängenden Speicher der Basisabbildung für die Ebene berechnet **90**. Anschließend werden die Texel für die gegenwärtige Ebene in die geeigneten Positionen in dem zusammenhängenden Speicher **92** kopiert.

[0031] An diesem Punkt wird in einem Schritt **94** ein Test durchgeführt, um zu bestimmen, ob die Zeigerflag auf "1" eingestellt war. Wenn sie es nicht war, wird wieder in die Schleife eingetreten, die oben beschrieben wurde (siehe **Fig. 3**), und zwar an einem Punkt "E". Wenn die Zeigerflag andererseits auf "1" eingestellt war, dann wird der Speicher für diese Ebene **96** freigemacht, dann wird die Zeigerflag in einem Schritt **97** auf "0" eingestellt, und dann wird der Speicherversatz in der gegenwärtigen Ebene gespeichert **98**. Anschließend wird wieder an einem Punkt "E" in die oben beschriebene Schleife eingetreten (siehe **Fig. 3**).

[0032] Wenn es alternativ in einem Schritt **74** bestimmt wird, daß die Ebenendaten nicht korrekt waren, und wenn die Zeigerflag auf "1" eingestellt ist, dann wird der Entscheidungskasten **76** anleiten, wieder in die oben beschriebene Schleife (siehe **Fig. 3**) an einem Punkt "E" einzutreten.

[0033] Wenn es andernfalls in einem Schritt **74** bestimmt wurde, daß die Ebenendaten nicht korrekt waren, und wenn die Zeigerflag auf "0" eingestellt ist, wird der Entscheidungskasten **76** bewirken, daß das Verfahren in einem Schritt **78** die Zeigerflag auf "1" einstellt, einen temporären Speicherblock für diese Ebene zuordnet **80** und bestätigt, daß der Speicher korrekt zugeordnet wurde **82**. Wenn bei der Speicherzuordnung ein Fehler auftrat, kann nichts weiter getan werden **84**. Alternativ werden die Daten dieser Ebene anschließend in den temporären Speicher kopiert, und es wird an dem Punkt "E" wieder in die oben beschriebene Schleife eingetreten (siehe **Fig. 3**).

[0034] Wieder bezugnehmend auf **Fig. 2** wird, wenn es in dem Schritt **32** bestimmt wurde, daß nicht die Basisebene vorhanden ist, in dem Verfahren zu dem Punkt "C" **36** weitergegangen (siehe **Fig. 5**). Alternativ würde zu dem Punkt "C" **36** weitergegangen, wenn in dem Schritt **32** die Basisebene existieren würde, wenn jedoch die Flag LevelOK in einem Schritt **34** nicht auf "0" gesetzt worden wäre.

[0035] Bezugnehmend nun auf **Fig. 5** besteht der nächste Schritt **13** von dem Punkt "C" **36** aus darin, zu bestimmen, ob die Ebeneninformationen korrekt sind. Wenn das nicht so ist, wird die Ebenengröße **15** berechnet, wird die Zeigerflag auf "1" eingestellt **17**, und wird ein temporärer Speicherblock für diese Ebene **19** zugeordnet. Anschließend wird bestätigt, daß der Speicher korrekt zugeordnet wurde **21**. Wenn bei der Speicherzuordnung ein Fehler auftrat, kann nichts weiter getan werden **23**. Alternativ wird zum Punkt "D" **58** gegangen.

[0036] An dem Punkt "C" **36** in dem Verfahren wird, wenn in dem Schritt **13** bestimmt wurde, daß die Ebeneninformationen korrekt waren, in dem Schritt **27** überprüft, ob die Zeigerflag auf "1" eingestellt ist. Wenn das der Fall ist, wird der temporäre Speicher für diese Ebene **28** freigemacht, wird die Zeigerflag in dem Schritt **31** auf "0" eingestellt, und wird ein Versatz für diese Ebene in dem zusammenhängenden Speicher **33** berechnet. Anschließend wird zu dem Punkt "D" **58** weitergegangen.

[0037] Wenn die Zeigerflag in dem Schritt **27** "0" war, dann muß nur ein Versatz für diese Ebene in dem zusammenhängenden Speicher **33** der Basisabbildung berechnet werden, bevor in dem Schritt **58** zu dem Punkt "D" übergegangen wird.

[0038] Wenn bezugnehmend auf **Fig. 3** die Schleife ihre fünfzehnte Iteration vollendet hat, dann wird in dem Schritt **60** der alte zusammenhängende Speicher freigemacht, wobei die Schleife bei **56** verlassen wird, und wobei das Verfahren in dem Schritt **58** bei "D" fortgesetzt wird.

[0039] Wenn an dem Punkt "D" angekommen wird, d. h. Schritt **58** des Verfahrens, werden Ebeneninformationen **39** gespeichert, wonach in dem Schritt **41** bestimmt wird, ob die Zeigerflag auf "1" eingestellt ist. Wenn dies der Fall ist, ist das Verfahren fertig **47**. Wenn die Zeigerflag auf "0" eingestellt wurde, dann wird in dem

Schritt **43** überprüft, ob der Versatz größer als "0" ist. Wenn der Versatz "0" ist, dann ist das Verfahren fertig **47**. Wenn die Zeigerflag "0" war, und wenn der Versatz größer als "0" ist, muß der Versatz für diese Ebene **45** gespeichert werden, wonach das Verfahren fertig ist **47**.

[0040] Unter Berücksichtigung des Verfahrens der vorliegenden Erfindung, wie es oben bezugnehmend auf die Fig. 2 bis Fig. 6 beschrieben wurde, und bezugnehmend auf den Code in der Sprache C im Anhang ist es nun möglich, spezifische Beispiele darzulegen, wie die Erfindung arbeitet. Bei jedem der folgenden Beispiele wird eine Herunterladesequenz gegeben sein, wobei jede Herunterladesequenz eine Ebenennummer, eine Ebenengröße (Breite × Höhe) und einen Zeigerflagwert für die Ebene umfassen wird.

Beispiel 1

Ebenennr.	Ebenengröße (B, H)	Zeigerflag
0	8 × 8	0
1	4 × 4	0
2	2 × 2	0
3	1 × 1	0

[0041] Im Beispiel 1 werden vier Ebenenabbildungen, die den Ebenen 0, 1, 2 und 3 entsprechen, heruntergeladen, wobei sie 8 × 8-, 4 × 4-, 2 × 2- bzw. 1 × 1-Abbildungen sind. Somit kann beim Herunterladen der Erste-Ebene-Abbildung für die Ebene 0 (d. h. die Basisabbildung) der gesamte Block des zusammenhängenden Speichers zugeordnet werden, und es existieren keine Probleme. Dies ist der einfachste Fall, da alle Ebenen heruntergeladen wurden, und zwar in der richtigen Reihenfolge, und da alle Ebenengrößen vollständig in Einklang miteinander sind.

Beispiel 2

Ebenennr.	Ebenengröße (B, H)	Zeigerflag
2	1 × 1	0
1	2 × 2	0
0	4 × 4	0

[0042] Im Beispiel 2 werden drei Ebenenabbildungen, die den Ebenen 2, 1 und 0 entsprechen, heruntergeladen, wobei dieselben 1 × 1-, 2 × 2- bzw. 4 × 4-Abbildungen sind. Somit kann beim Herunterladen der Erste-Ebene-Abbildung für die Ebene 2 der gesamte Block des zusammenhängenden Speichers zugeordnet werden, wobei beim weiteren in Einklang stehenden Herunterladen der Ebenen 1 und 0 zu sehen ist, daß keinen Problemen begegnet werden wird. Wieder waren alle Ebenennummer und Ebenengrößen vollständig in Einklang miteinander.

Beispiel 3

Ebenenur.	Ebenengröße (B, H)	Zeigerflag
0	4 × 4	0
1	16 × 16	1
2	1 × 1	0
1	2 × 2	0

[0043] Bei dem Beispiel 3 wurden vier Ebenenabbildungen, die den Ebenen 0, 1, 2 und 1 entsprechen, heruntergeladen, und dieselben sind jeweils eine 4 × 4-, eine 16 × 16-, eine 1 × 1- bzw. 2 × 2-Abbildungen. Somit wurde beim Herunterladen der Erste-Ebene-Abbildung für die Ebene 0 (d. h. die Basisabbildung) der gesamte Block des zusammenhängenden Speichers zugeordnet, und es wurde angenommen, daß zusätzliche Ebenenabbildungen empfangen werden, wobei die Ebene 1 eine 2 × 2-Abbildung, die Ebene 2 eine 4 × 4-Abbildung, usw. sein würden. Es sei angemerkt, daß nicht bekannt ist, wieviele Abbildungen empfangen werden. Bei dem Herunterladen einer Ebene-1-16 × 16-Abbildung ist jedoch zu erkennen, daß eine Inkonsistenz be-

steht. Demgemäß wird die Ebene-1-16 × 16-Abbildung in den temporären Speicher plaziert, wonach die Zeigerflag auf 1 eingestellt wird, was anzeigt, daß ein Problem vorhanden ist. Das Problem löst sich selbst, wenn zusätzliche Herunterladevorgänge, die mit der ursprünglichen Ebene 0 in Einklang sind, heruntergeladen werden (d. h. Ebene 2 mit 1 × 1 und Ebene 1 mit 2 × 2). Demgemäß kann die ursprünglich heruntergeladene "Ebene 1" mit 16 × 16 weggeworfen werden, und der temporäre Speicher kann freigemacht werden. Es sei angemerkt, daß der ursprüngliche zusammenhängende Speicherblock bei diesem Beispiel basierend auf der ersten heruntergeladenen Ebene verwendet wurde.

Beispiel 4

Ebenennr.	Ebenengröße (B, H)	Zeigerflag
0	8 × 8	0
1	2 × 2	1
2	1 × 1	1
0	4 × 4	0

[0044] Im Beispiel 4 werden vier Ebenenabbildungen, die den Ebenen 0, 1, 2 und 0 entsprechen, heruntergeladen, wobei dieselben eine 8 × 8-, eine 2 × 2-, eine 1 × 1- bzw. 4 × 4-Abbildung sind. Bei dem Herunterladen der Erste-Ebene-Abbildung für die Ebene 0 (d. h. die Basisabbildung) wurde ein gesamter Block des zusammenhängenden Speichers zugeordnet. Würde jedoch als nächste Ebene die Ebene 1 heruntergeladen werden, und hätte sie eine Größe, die mit der Ebene-0-Größe nicht in Einklang ist, müßten die Ebene-1-(2 × 2-)Daten in den temporären Speicher plaziert werden, und die Zeigerflag würde auf "1" eingestellt. Wenn die dritte Ebene, d. h. die Ebene 2, als eine 1 × 1-Abbildung heruntergeladen werden würde, wäre dieselbe ebenfalls mit der Größe der ursprünglichen Ebene 0 nicht in Einklang, weshalb sie ebenfalls in dem temporären Speicher plaziert werden würde, und die Zeigerflag auf "1" eingestellt werden würde. Wenn schließlich die vierte Ebene sich selbst als (eine neue) Ebene 0 identifizieren würde, und wenn dieselbe mit den anderen heruntergeladenen Ebenen 1 und 2 konsistent sein würde, könnten die neue Ebene 0 und die vorher heruntergeladenen Ebenen 1 und 2 in dem zusammenhängenden Speicher gemäß der vorliegenden Erfindung plaziert werden.

[0045] Wie es für Fachleute offensichtlich ist, liefert das vorliegende erfindungsgemäße Verfahren eine Einrichtung zum Zuordnen eines zusammenhängenden Speicherblocks für alle Ebenen beim Empfang einer Abbildung einer beliebigen Ebene. Wenn Abbildungen zusätzlicher Ebenen heruntergeladen werden, schafft die vorliegende Erfindung ein Verfahren zum Bestätigen, daß sie mit der früheren Zuordnung in Einklang sind, oder das vorliegende Verfahren speichert alternativ ihre Informationen vorübergehend, bis ein in Einklang stehender Satz von Ebenenabbildungen heruntergeladen ist. Bei der Verifikation des Empfangs eines in Einklang stehenden Satzes von Ebenenabbildungen werden alle in Einklang stehenden Ebenenabbildungsdaten in einem einzigen zusammenhängenden Speicherblock sein.

Anhang-Quellencode

[0046] /*

ERKLÄRUNG DER BEGRENZTEN RECHTE

[0047] Eine Verwendung, Duplizierung und eine Offenbarung durch die U. S. Regierung ist Begrenzungen unterworfen, wie sie in der Unterteilung (b) (3) (ii) der Bestimmung zu Rechten auf technische Daten und Computersoftware bei 52.227-7013 dargelegt ist.

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include "mem_alloc.h"
#include "GL/gl.h"
#include "ogl_types.h"
#include "ogl_state.h"
#include "ogl_texture.h"
#include "ogl_env.h"
#include "ds_public.h"
#include "ocm_cmd.h"
#include "pcm_cmd.h"

/*
**********************************************************************
**
```

```
** Zweck: Berechnen des Speichers, der für eine volle MIP-
** Abibldung benötigt wird.
**
** Eingaben: w        - Breite der Abbildung
**           h        - Höhe der Abbildung
**           myBorder - Randgröße für die Abbildung
**           tSize    - Texelgröße für die interne Speiche-
**                      rung
**           level    - Ebenennummer
**           mipLevel - gibt die Gesamtanzahl von MIP-Ebenen
**                      minus 1 zurück
**           offset   - wird verwendet, um den berechneten
**                      Versatz zurückzugeben
**
** Gibt die berechnete Speichergröße zurück.
**
*****************************************************************
*/

static Int32 computeMipMemSize(
   Int32 w,
   Int32 h,
   Int32 myBorder,
   Int32 tSize,
   Int32 level,
   Uint16 *mipLevel,
   Int32 *offset)
{
   int bSize;
   int memSize;
   int numLevels;
   int high;

   /*
   * Dies wird die Größe der vollen MIP-Abbildung berechnen
   * (bis herunter zu 1x1).
   */
```

```
    memSize = 0;
    high = (w > h) ? w : h;

    for (numLevels = 0; high >= 1; high >>= 1)
    {

       /*
       * Speichern des Versatzes der gegenwärtigen  Ebene  in
       * dem Speicher.
       */

       if(numLevels == level)
          *offset = memSize;

       /*
       * Berechnen des Speichers für die gegenwärtige Ebene.
       */

    bSize = (2 * myBorder) * w + (2 * myBorder) * h +
            (myBorder * myBorder) * 4;
    memSize += tSize * (w * h + bSize);

       w = (w + 1) / 2;
       h = (h + 1) / 2;

       numLevels++;
    }

    *mipLevel = numLevels;
    return(memSize);
}

/*
************************************************************
**
**  Zweck: Überprüfen, um zu sehen, ob die gegenwärtige Ebe-
**         ne der Abbildung die gleiche  Breite,  Höhe (2D),
**         Randgröße und Format wie die Basisabbildung hat.
```

```
**
** Eingaben: target - Dimension der Texturabbildung
**           mipLevel - Basisebeneninformationen
**           width  - berechnete Breite für  die Basisabbil-
**                    dung, die der gegenwärtigen Ebene ent-
**                    spricht
**           height - berechnete Höhe  für  die  Basisabbil-
**                    dung, die der gegenwärtigen Ebene ent-
**                    spricht
**           border - Randgröße der gegenwärtigen Ebene
**           internalFormat - Benutzer  gab  das  interne
**                            Format der gegenwärtigen Ebene
**                            ein
**
** Gibt -1 zurück, wenn Ebene Ok, sonst 0
**
*************************************************************
*/

static Int32 checkLevel(
   Enum target,
   Int32 level,
   MipLevelState mipLevel,
   Uint32 width,
   Uint32 height,
   Uint32 border,
   Enum internalFormat)
{
   int levelOk;
   Uint32 tmpW;
   Uint32 tmpH;

   levelOk = 1;

   tmpW = mipLevel.width >> level;
   if(tmpW == 0)
      tmpW = 1;
   if(tmpW != width)
```

```
      levelOk = 0;

   if(target == GL_TEXTURE_2D)
   {
       tmpH = mipLevel.height >> level;
       if(tmpH == 0)
          tmpH = 1;
       if(tmpH != height)
          levelOk = 0;
   }

if(mipLevel.border != border)
   levelOk = 0;

   if(mipLevel.internalFormat != internalFormat)
      levelOk = 0;

   return(levelOk);
}

/*
**********************************************************
**
** Zweck: Berechnen des Speichers,  der  für eine MIP-Abbil-
**        dungsebene benötigt wird.
**
** Eingaben: width    - Breite der Basisebene
**           height   - Höhe der Basisebene
**           tSize    - Texelgröße für die interne  Speiche-
**                      rung
**           myBorder - Größe  des  Rands  für  die  interne
**                      Speicherung
**
** Gibt die berechnete Ebenengröße zurück
**
**********************************************************
*/
```

```
Int32_hpOcm_computeLevelSize(
   Int32width,
   Int32 height,
   Int32 tSize,
   Int32 myBorder)
{
   int bSize;
   int levelSize;

   bSize = (2 * myBorder) * width + (2 * myBorder) *
           height + (myBorder * myBorder) * 4;
   levelSize = tSize * (width * height + bSize);

   return(levelSize);
}

/*
**************************************************************
**
** Zweck: Berechnen des Texelversatzes von  der  Basisabbil-
**        dung für die gegenwärtige Ebene.
**
** Eingaben: width    - Breite der Ebene
**           height   - Höhe der Basisebene
**           tSize    - Texelgröße für die interne  Speiche-
**                      rung
**           myBorder - Größe  des  Rands  für  die  interne
**                      Speicherung
**           Level    - Ebene der MIP-Abbildung
**
** Gibt den berechneten Versatz zurück.
**
**************************************************************
*/

Uint32_hpOcm_computeTexelOffset(
   Int32 width,
   Int32 height,
```

```
    Int32 tSize,
    Int32 myBorder,
    Int32 level)
{
    int i;
    unsigned int offset;

    offset = 0;

    /*
    * Summieren  der Größe jeder Ebene, deren Nummer  kleiner
    * als die gegenwärtige Ebene ist.
    */

    for(i=0; i<level; i++)
    {
        offset += _hpOcm_computeLevelSize(width, height,
                  tSize, myBorder);

        width  = (width  + 1) / 2;
        height = (height + 1) / 2;
    }

    return(offset);
}

/*
*****************************************************************
**
** Zweck: Zuordnen einer internen Speicherung für  Texturab-
**        bildungen.
**
** Eingaben: memSize - Größe des zuzuordnenden Stücks
**           *sharedID - zum Zurückgeben der gemeinsam  ver-
**                       wendeten  Speicher-ID,   wenn  eine
**                       vorhanden ist
**           *ptr - wird  verwendet,   um  den  zugeordneten
**                  Speicherzeiger zu speichern
```

```
**          *sharedTexels - Flag, die anzeigt, ob ein ge-
**                          meinsam verwendeter Speicher
**                          zugeordnet wurde
**          targetIndex - Dimension der Abbildung
**
** Gibt -0 zurück, wenn erfolgreich
** Gibt -1 zurück, wenn der Speicher nicht ausreicht
**
** Umgebungsvariablen: OGL_TXTR_SHMEM_THRESHOLD
**                     Diese Umgebungsvariable setzt einen
**                     Zaun für die Verwendung von Prozeß-
**                     speicher als Funktion von gemeinsam
**                     verwendetem Speicher. Jede Textur,
**                     die eine Größe > die Schwelle hat,
**                     wird in dem gemeinsam verwendeten
**                     Speicher gespeichert. Der Anfangswert
**                     ist auf 1.024 * 1.024 (Bytes) einge-
**                     stellt.
**
** Algorithmus: Wenn der angeforderte Speicher größer als
**              der gemeinsam verwendete Speicherzaun ist,
**              Zuordnen des Speichers aus dem gemeinsam
**              verwendeten Speichervorrat. Sonst
**              Zuordnen des regulären Speichers.
**
*************************************************************
*/

Int32 allocateTexelMemory(
   Uint32 memSize,
   Uint32 *sharedID,
   Texel *ptr,
   unsigned *sharedTexels,
   Enum targetIndex)
{
   int shmID;
   char *tmpPtr;
   char *cp;
```

```
long int tmp_shared_memory_limit;

if(memSize == 0)
{
    ptr->lum8 = (TexelLum8 *)NULL;
    *sharedTexels = 0;
    return(0);
}

/*
* Mögliche Optimierung: Speichern des Werts dieser  Umge-
*                       bungsvariable   ist  ein  PCM-Zu-
*                       stand.  Dann  existiert  nur  ein
*                       Getenv.
*/

cp = OglGetenv(HPOGLINT_TXTR_SHMEM_THRESHOLD);

if(cp)
{
    tmp_shared_memory_limit = (long int) atol(cp);

    /*
    * Klemmen von negativen auf eins...
    */

    if(tmp_shared_memory_limit < 1)
       tmp_shared_memory_limit = 1L;
}
else

    tmp_shared_memory_limit = OGL_TXTR_SHMEM_DEFAULT;
shmID = -1;

/*
* Zuordnen eines gemeinsamen Speichers,  wenn  die  Spei-
* chergröße größer als die Schwelle ist,  oder  wenn dies
* keine eindimensionale Abbildung ist.  Es  werden  keine
```

```
* eindimensionalen Abbildungen durchgelassen, da das For-
* mat  der  Abbildung unterschiedlich als das  verwendete
* ist.
*/

if((memSize >= tmp_shared_memory_limit) &&
   (targetIndex != OGL_TEXTURE_1D))
{
   shmID = shmget(IPC_PRIVATE, memSize, IPC_CREAT0666);
   if(shmID != -1)
   {
      tmpPtr = (char*)shmat(shmID, (char*)0, 0);
      if(int)tmpPtr == -1)
      {
         /*
         * Entfernen der ID des gemeinsam verwendeten
         * Speichers (Shared Memory ID).
         */

         shmctl(shmID, IPC_RMID, 0);
         shmID = -1;
      }
      sonst
      {
         /*
         * Es existiert eine gültige Shared Memory ID und
         * es ist Speicher angebracht.  Somit werden die
         * shmID und der Speicherzeiger gespeichert, und
         * es wird 1 zurückgegeben, was bedeutet, daß der
         * gemeinsam verwendete Speicher erfolgreich
         * eingesetzt wurde.
         */

         *sharedID = shmID;
         ptr->lum8 = (TexelLum8*)tmpPtr;
         *sharedTexels = 1;
         return(0); /* Shared Memory Success! */
      }
```

```
      }
   }

   if(shmID == -1)
   {
      /*
      * Versucht, regelmäßigen Speicher zuzuordnen.
      */

      tmpPtr = (char*)SumMalloc(memSize, FREE_MANUALLY);
      if(tmpPtr == (char*)NULL)
      {
         /*
         * Kein Speicher, keine Texturabbildung.
         */
         SET_OGL_ERROR(GL_OUT_OF_MEMORY);
         return(-1);
      }
      else
      {
         /*
         * Es existiert Speicher. Somit soll der Speicher-
         * zeiger gespeichert werden, und es wird 0 zurück-
         * gegeben, da der reguläre  Speicher  erfolgreich
         * verwendet wurde.
         */

         ptr->lum8 = (TexelLum8*)tmpPtr;
         *sharedTexels = 0;
         return(0); /* Erfolg für regulären Speicher! */
      }
   }
   return(0);
}

/*
**********************************************************
**
```

```
** Zweck: Freimachen eines internen Texelspeichers.
**
** Eingaben: sharedTexel - Flag, die gemeinsam verwendeten
**                         Speicher anzeigt
**           sharedID - ID des gemeinsam verwendeten Spei-
**                      chers, wenn er anwendbar ist
**           *ptr - Speicherzeiger
**
*************************************************************
*/

void_hpOcm_freeTexelMemory(
   unsigned sharedTexel,
   Uint32 sharedID,
   void *ptr)
{
   if(sharedTexel)
   {
      shmdt(ptr);
      shmctl(sharedID, IPC_RMID, 0);
   }
   else
   {
      if(ptr != (void*)NULL)
         SumFree(ptr);
   }
}

#pragma inline Log2
static int Log2(int Value)
{
   int Result = 0;
   while(Value > 1){
      Result++, Value >>= 1;
   }
   return Result;
}
```

```
/*
************************************************************
**
** Zweck: Diese Routine ordnet den internen Graphikkernspei-
**        cher für Texturabbildungen zu, führt eine zweidi-
**        mensionale Pipeline aus, um die Pixel aufzupacken
**        und zu übertragen, und verfolgt die Anzahl von
**        gültigen MIP-Abbildungsebenen, die für jede Tex-
**        turabbildung heruntergeladen wurden.
**
**        Diese Routine ist komplex, da sie alle Ebenen der
**        MIP-Abbildung in einem zusammenhängenden Speicher
**        hält.
**
** Eingaben: target - spezifiziert GL_TEXTURE_1D oder
**                    GL_TEXTURE_2D
**           level - Detailebenen-Nummer
**           internalFormat - Benutzer-spezifiziertes,
**                            internes Format
**           width - Breite der Textur  ** ENTHÄLT NICHT DEN
**                   RAND **
**           height - Höhe der  Textur  ** ENTHÄLT NICHT DEN
**                    RAND **
**           border - Texturabbildungsrandgröße
**
** Wird von glTexImage*D aufgerufen
**
** Global zugegriffene Variablen - _hpOgl_context
**
** Algorithmus: Wenn die Basisabbildung keinen Speicher hat,
**              Zuordnen von genug Speicher für eine MIP-Ab-
**              bildung. Wenn das Format oder die Randgröße
**              der ankommenden Ebene nicht mit der ge-
**              genwärtigen Basisebene konsistent ist, Zu-
**              ordnen von Speicher für die Ebene getrennt
**              von dem Rest der Abbildung und Markieren
**              der Ebene als inkonsistent.
**
```

```
** Aufgerufene Funktionen: getInternalFormatAndSize
**                          computeMipMemSize
**                          allocateTexelMemory
**                          checkLevel
**                          DS_NO_LOCK.destroyTexture
**                          _hpOcm_computeLevelSize
**                          _hpOcm_computeTexelOffset
**                          memcpy
**                          SumFree
**                          SumMalloc
**                          _hpOcm_freeTexelMemory
**                          storeMagicBorderValue
**
** Fehlernachrichten: GL_OUT_OF_MEMORY
**
*************************************************************
*/

static void downLoadTexels(
   Enum target,
   Int32 level,
   Int32 internalFormat,
   Int32 width,
   Int32 height,
   Int32 border)
{

   TextureObjectPtr boundTexels;
   MipLevelStatePtr baseMap;
   MipLevelStatePtr thisLevel;
   Int32 offset;
   Int32 tSize;
   Int32 levelOk;
   Int32 ourType;
   Uint32 memSize;
   int i;
   int myBorder;
   unsigned tmpSharedTexels;
```

```
Enum targetIndex;
int32 ignore[4];

/*
* Setze myBorder auf 1, da immer Raum für einen Rand  zu-
* geordnet wird,
* selbst wenn der Benutzer keinen  Rand  mit  den  Texeln
* hat.
* Initialisieren von Versatz auf 0.
*/

myBorder = INTERNAL_BORDER_SIZE;
offset = 0;

/*
* Erhalten eines Zeigers auf  die  gegenwärtig  begrenzte
* Textur mit korrekter Dimension.
* Einstellen von baseMap und thisLevel.
*/

if(target == GL_TEXTURE_2D)
   targetIndex = OGL_TEXTURE_2D;
else
{
   targetIndex = OGL_TEXTURE_1D;
   if(width > 0)
      height = 1;
   sonst
      height = 0;
}

boundTexels = OGL_TEXOBJ.boundTextures[targetIndex];

baseMap   = &boundTexels->level[0];
thisLevel = &boundTexels->level[level];

/*
* Wenn die gegenwärtig begrenzte Textur  ein  Teil  einer
```

```
* Anzeigeliste ist
* Optimierung, Kopieren der Textinformationen auf einen
* neuen Zeiger, derart, daß der Anzeigelistenzeiger nicht
* verändert wird.
*/

if(boundTexels->hdr.dlmFlag == 1)
{
   Texel texelSave;

   texelSave.lum8 = baseMap->texelData.lum8;

   memSize = computeMipMemSize(baseMap->width,
             baseMap->height,
             myBorder
             baseMap->bytesPerTexel,
             0,
             &boundTexels->hdr.numMipLevels,
             &offset);

   if(-1 == allocate TexelMemory(memSize,
            &boundTexels->hdr.sharedTexelID,
            &baseMap->texelData,
            &tmpSharedTexels,
            boundTexels->targetIndex))
   {
      /*
      * Fehler wurde bereits gesetzt.
      */

      return;
   }

   boundTexels->hdr.sharedTexels = tmpSharedTexels;
   baseMap->pointerFlag = !TMP_MEMORY_POINTER;

   memcpy(baseMap->texelData.lum8,        texelSave.lum8,
   memSize);
```

```
   boundTexels->hdr.dlmFlag = 0;
}

DS_NO_LOCK.getInternalFormatAndSize(internalFormat,
&ourType, &tSize, ignore);

/*
* Überprüfen nach  einem  existierenden  Texelabbildungs-
* speicher.
* Das Flußdiagramm der Fig. 2 bis 6 beginnt hier.
*/

if(baseMap->texelData.lum8 == (TexelLum8*)NULL)
{
   Uint32 wSave;
   Uint32 hSave;

   /*
   * Berechnen der Größe der Basisebene.  Diese  Informa-
   * tionen werden verwendet, um sicherzustellen, daß die
   * gegenwärtige Ebene eine korrekte MIP-Ebene ist, und/
   * oder dieselben werden zur Speicherzuordnung  verwen-
   * det.
   */

   wSave = width;
   hSave = height;

   if(level != 0)
   {
   wSave <<= level;
   if(target == GL_TEXTURE_2D)
      hSave <<= level;
   }

   /*
   * Berechnen des gesamten Speichers, der für die Texel-
```

```
    * daten benötigt wird.
    * Dieser Code ordnet genug Speicher für eine volle
    * MIP-Abbildung einschließlich der Ränder (mit der
    * Größe 1) zu.
    */

    memSize = computeMipMemSize(wSave,  hSave,  myBorder,
              tSize, level,
              &boundTexels->hdr.numMipLevels,
              &offset);

    if(-1 == allocateTexelMemory(memSize,
             &boundTexels->hdr.sharedTexelID,
             &baseMap->texelData,
             &tmpSharedTexels,
             targetIndex))
    {
       /*
       * Fehler wurde bereits eingestellt.
       */

       return;
    }

    boundTexels->hdr.sharedTexels = tmpSharedTexels;

baseMap->texelType = ourType;
baseMap->internalFormat = internalFormat;
baseMap->bytesPerTexel = tSize;
baseMap->border = border;
baseMap->width = wSave;
baseMap->height = hSave;
  baseMap->widthLog2 = Log2(wSave);
  baseMap->heightLog2 = Log2(hSave);
baseMap->pointerFlag = !TMP_MEMORY_POINTER;
baseMap->depth = 0;
  baseMap->depthLog2 = 0;
```

```
}

/*
* Der Speicher ist nun zugeordnet, oder er existierte be-
* reits. Nun wird die Ebene mit den Texelinformationen
* gefüllt.
*
* Überprüfen, um sicherzustellen, daß diese Ebene eine
* gültige MIP-Ebene ist. Dies umfaßt das Aufweisen der
* gleichen Randgröße und des internen Formats wie die Ba-
* sisebene sowie das Aufweisen eines korrekten Breiten-
* werts und eines korrekten Höhenwerts (zweidimensional).
*/

levelOk = checkLevel(target, level, *baseMap, width,
                     height, border, internalFormat);

/*
* Wenn die Textur begrenzt ist, übermitteln der DSM
* (Hardware), daß diese Texel verändert werden.
*/

if((boundTexels->hdr.texelID != INVALID_TEXEL_ID) &&
   (!levelOk))
   DS_NO_LOCK.destroyTexture(boundTexels);

if(level == 0)
{
   if(!levelOk)
   {
      /*
      * Es wurde bereits genug Speicher zugeordnet, um
      * Texel zu speichern, wobei sich jedoch
      * die Basisabbildungsgröße verändert hat. Deshalb
      * wird der alte Speicherzeiger gespeichert, wird
      * ein neuer Speicher zugeordnet, werden die Ebenen
      * der alten Abbildung, die gültig sind, in die neue
      * Abbildung kopiert, wird der Rest der Ebenen zu
```

```
   * den tmp-Zeigern kopiert, wonach der alte Speicher
   * freigemacht wird.
   */

   Texel texelSave;
   unsigned sharedTexelsSave;
   Uint32 sharedTexelIDSave;

   /*
   * Einfüllen von neuen Werten in die Basisebene.
   */

baseMap->texelType = ourType;
baseMap->internalFormat = internalFormat;
baseMap->bytesPerTexel = tSize;
baseMap->border = border;
baseMap->width = width;
baseMap->height = height;
  baseMap->widthLog2 = Log2(width);
  baseMap->heightLog2 = Log2(height);
baseMap->pointerFlag = !TMP_MEMORY_POINTER;
baseMap->depth = 0;
baseMap->depthLog2 = 0;

   /*
   * Speichern des gegenwärtigen Texelzeigers, der
   * Shared Memory ID und der Shared Memory Flag.  Wir
   * brauchen diese Dinge, um den  Speicher  freizuma-
   * chen, wenn wir fertig sind.
   */

   texelSave.lum8 = baseMap->texelData.lum8;
   sharedTexelsSave = boundTexels->hdr.sharedTexels;
   sharedTexelIDSave = boundTexels->hdr.sharedTexelID;

   memSize = computeMipMemSize(width,
             height,
             myBorder, tSize,
```

```
            level,
            &boundTexels->hdr.numMipLevels,
            &offset);

if(-1 == allocateTexelMemory(memSize,
            &boundTexels->hdr.sharedTexelID,
            &baseMap->texelData,
            &tmpSharedTexels,
            targetIndex))

{
   /*
   * Fehler wurde bereits eingestellt.
   */

   return;
}

boundTexels->hdr.sharedTexels = tmpSharedTexels;

/*
* Für jede Ebene in der existierenden Abbildung,
* Bestimmen der Ebenen, die nun "ok" sind, und Ko-
* pieren dieser Ebenen in die neue Abbildung. Ko-
* pieren des Rests der Ebenen zu tmp-Zeigern und
* Setzen der Zeigerflag.
*/

for(i=1; i<OGL_MAX_MIPMAP_LEVELS; i++)
{
   int levelSize;
   MipLevelStatePtr curLevel;

   curLevel = &boundTexels->level[i];

   if(curLevel->texelData.lum8 != NULL)
   {
      levelSize = _hpOcm_computeLevelSize(curLevel
```

```
                ->width, curLevel->height,
                curLevel->bytesPerTexel,
                myBorder);

levelOk = checkLevel(target,
                     i,
                     *baseMap,
                     curLevel->width,
                     curLevel->heigth,
                     curLevel->border,
                     curLevel->internalFormat);
if(levelOk)
   {
      offset = _hpOcm_computeTexelOffset(baseMap
               ->width,
               baseMap->height,
               baseMap->bytesPerTexel,
               myBorder, level);

memcpy(baseMap->texelData.lum8 + offset,
       curLevel->texelData.lum8,
       levelSize);

   if(curLevel->pointerFlag==TMP_MEMORY_POINTER)
     {
      SumFree(curLevel->texelData.lum8);
      curLevel->pointerFlag=!TMP_MEMORY_POINTER;
     }

curLevel->texelData.lum8=baseMap->texelData.lum8
                         + offset;
   }
   else
   {
      Texel levelSave;

      /*
      * Wenn diese Ebene die Zeigerflag  bereits
```

```
            * gesetzt hat, dann  besteht  kein  Grund,
            * die Ebene zu kopieren.
            */

            if(curLevel->pointerFlag != TMP_MEMORY_
                                        POINTER)
            {
            levelSave.lum8=curLevel->texelData.lum8;
            curLevel->pointerFlag=TMP_MEMORY_POINTER;
            curLevel->texelData.lum8 =
                      (TexelLum8*)SumMalloc(levelSize,
                      FREE_MANUALLY);

            if(curLevel->texelData.lum8==(TexelLum8*)
                                        NULL)

               {
                  SET_OGL_ERROR(GL_OUT_OF_MEMORY);
                  return;
               }

         memcpy(curLevel->texelData.lum8,
                levelSave.lum8, levelSize);
            }
         }
      }
   }

   /*
   * Freimachen des alten Speichers in dem  texelSave-
   * Zeiger.
   */

   _hpOcm_freeTexelMemory(sharedTexelsSave,
                          sharedTexelIDSave,
                          (void*)texelSave.lum8);

}
```

```
}
sonst
{
   /*
   * Die weitergeleitete Ebene ist nicht die Basisebene.
   */

   if(levelOk)
   {
      if(thisLevel->pointerFlag == TMP_MEMORY_POINTER)
      {
         /*
         * Die Ebene ist nun Ok, wenn sie es vorher  noch
         * nicht war. Deshalb  Freimachen  des  tmp-Spei-
         * chers, Rücksetzen der Zeiger-Flag  und  Neube-
         * rechnen des Versatzes in  dem  mip-Array,  auf
         * den durch die Basisabbildung gezeigt wird.
         */

         SumFree(thisLevel->texelData.lum8);
         thisLevel->pointerFlag = !TMP_MEMORY_POINTER;
         offset = _hpOcm_computeTexelOffset(baseMap
                  ->width,
                  baseMap->height,
                  baseMap->bytesPerTexel,
                  myBorder, level);
      }
      else
      {
         /*
         * This level...
         *
         *
         *
         */
         offset = _hpOcm_computeTexelOffset(baseMap
                  ->width,
                  baseMap->height,
```

```
                    baseMap->bytesPerTexel,
                    myBorder, level);
      }
   }
   sonst
   {
      /*
      * Ebene ist nicht Ok.  Somit,  Zuordnen  eines  be-
      * stimmten temporären Speichers und Speichern  die-
      * ser Ebenentexel dort. Ebenfalls,  Einstellen  der
      * Zeiger-Flag, um anzuzeigen, daß die Ebene  keinen
      * Versatz zur Basisabbildung aufweist.
      */

   int levelSize;

   levelSize = _hpOcm_computeLevelSize(width,     height,
               tSize, myBorder);

   thisLevel->pointerFlag = TMP_MEMORY_POINTER;
   thisLevel->texelData.lum8=(TexelLum8*)SumMalloc
                             (levelSize, FREE_MANUALLY);

   if(thisLevel->texelData.lum8 == (TexelLum8*)NULL)
     {
      SET_OGL_ERROR(GL_OUT_OF_MEMORY);
      return;
     }
   }
}

/*
* Speichern der Ebeneninformationen.
*/

thisLevel->texelType = ourType;
thisLevel->internalFormat = internalFormat;
thisLevel->bytesPerTexel = tSize;
```

```
   thisLevel->border = border;
   thisLevel->width = width;
   thisLevel->height = height;
   thisLevel->widthLog2 = Log2(width);
   thisLevel->heightLog2 = Log2(height);
   thisLevel->depth = 0;
   thisLevel->depthLog2 = 0;

   /*
   * Wenn diese Ebene Ok ist, dann Speichern des Versatzes
   * für diese Ebene in den Texeldaten. Es sei angemerkt:
   * wenn diese die Ebene 0 ist, dann wird der Versatz immer
   * 0 sein.
   */

   if(thisLevel->pointerFlag != TMP_MEMORY_POINTER)&&offset)
      thisLevel->texelData.lum8 = baseMap->texelData.lum8 +
      offset;
   if(thisLevel->texelData.lum8)
      storeMagicBorderValue(tSize, (char*)thisLevel->
      texelData.lum8);
}
```

Patentansprüche

1. Verfahren zum Speichern von Texeldaten einer Textur in einem zusammenhängenden Speicherblock eines Speichers einer Texturabbildungshardware in einem Computergraphiksystem, wobei einer Textur eine MIP-Abbildung mit einer Mehrzahl von MIP-Abbildungsebenen (**100**, **102**, **104**, **108**) zugeordnet ist, wobei das Verfahren folgende Schritte aufweist

(a) Empfangen (**12**) von Texeldaten für eine erste heruntergeladene MIP-Abbildungsebene (**100**, **102**, **104**, **108**);

(b) Berechnen (**16**) eines zusammenhängenden Speicherblocks für eine vollständige MIP-Abbildung, basierend auf der Größe und der Ebenennummer der ersten heruntergeladenen MIP-Abbildungsebene (**100**, **102**, **104**, **108**);

(c) Zuordnen (**18**) des zusammenhängenden Speicherbocks für die vollständige MIP-Abbildung;

(d) Bestimmen (**33**) eines Versatzwertes in dem zusammenhängenden Speicherblock, basierend auf der Ebenennummer der ersten heruntergeladenen MIP-Abbildungsebene (**100**, **102**, **104**, **108**), wobei der Versatzwert der vorbestimmten Position zum Speichern von Texeldaten entspricht, die der ersten heruntergeladenen MIP-Abbildungsebene (**100**, **102**, **104**, **108**) zugeordnet sind;

(e) Speichern (**39**) der Texeldaten für die erste heruntergeladene MIP-Abbildungsebene (**100**, **102**, **104**, **108**) an der Position in dem zusammenhängenden Speicherblock, auf den durch den Versatzwert gezeigt wird;

(f) wiederholtes Empfangen zusätzlicher Texeldaten für zusätzliche MIP-Abbildungsebenen (**100**, **102**, **104**, **108**), nachdem die Texeldaten für die erste heruntergeladene MIP-Abbildungsebene (**100**, **102**, **104**, **108**) empfangen worden sind, und für alle folgenden Texeldaten:

(1) Bestätigen (**28**), dass die neu empfangenen Texeldaten für die zusätzliche MIP-Abbildungsebene (**100**, **102**, **104**, **108**) der MIP-Abbildung zugeordnet sind, der die erste heruntergeladene MIP-Abbildungsebene (**100**, **102**, **104**, **108**) zugeordnet ist, und wenn dies der Fall ist, Berechnen (**33**) eines Versatzes in dem zusammenhängenden Speicherblock und Plazieren der zusätzlichen Texeldaten an der Position in dem zusammenhängenden Speicherblock, auf die durch den Versatz gezeigt wird,

wenn dies nicht der Fall ist, Plazieren (**29**) der zusätzlichen heruntergeladenen Texeldaten in einem temporä-

ren Speicher, und Einstellen (**31**) einer Zeigerflag auf einen Wert, der anzeigt, dass sich Texeldaten in dem temporären Speicher befinden;
(2) Wiederholen des Schritts (e) (1), bis alle zum Speichern bereitgestellten Texeldaten heruntergeladen sind;
(3) falls in dem zusammenhängenden Speicherblock Texeldaten für eine MIP-Abbildung gespeichert sind, Entfernen der in dem temporären Speicher befindlichen Texeldaten; und
(4) falls in dem zusammenhängenden Speicherblock keine Texeldaten für eine MIP-Abbildung gespeichert sind und falls in dem temporären Speicher Texeldaten für eine MIP-Abbildung gespeichert sind, Platzieren der in dem temporären Speicher befindlichen Texeldaten in dem zusammenhängenden Speicher.

2. Verfahren nach Anspruch 1, bei dem im Schritt (a) ein erster Datensatz empfangen wird, der der ersten heruntergeladenen MIP-Abbildungsebene (**100**, **102**, **104**, **108**) entspricht, und bei dem der Schritt (f)(1) folgende Schritte umfaßt:
Analysieren des ersten Datensatzes, um eine Größe und eine Ebenennummer des ersten Datensatzes zu bestimmen;
Empfangen eines zweiten Datensatzes, der der zusätzlichen MIP-Abbildungsebene (**100**, **102**, **104**, **108**) entspricht;
Analysieren des zweiten Datensatzes, um eine Größe und eine Ebenennummer des zweiten Datensatzes zu bestimmen;
Vergleichen der Größe und der Ebenennummer des zweiten Datensatzes mit der Größe und der Ebenennummer des ersten Datensatzes; und
auf der Grundlage der Größe und der Ebenennummer des zweiten Datensatzes und auf der Grundlage des Vergleichsschrittes, Bestimmen, ob die neu empfangenen Texeldaten für die zusätzliche MIP-Abbildungsebene (**100**, **102**, **104**, **108**) der MIP-Abbildung zugeordnet sind, der die erste heruntergeladene MIP-Abbildungsebene (**100**, **102**, **104**, **108**) zugeordnet ist.

3. Verfahren nach Anspruch 2, das ferner den Schritt des Bestimmens einer Anzahl von MIP-Abbildungsebenen (**100**, **102**, **104**, **108**) basierend auf der Größe und der Ebenennummer des ersten Datensatzes umfaßt.

Es folgen 6 Blatt Zeichnungen

Anhängende Zeichnungen

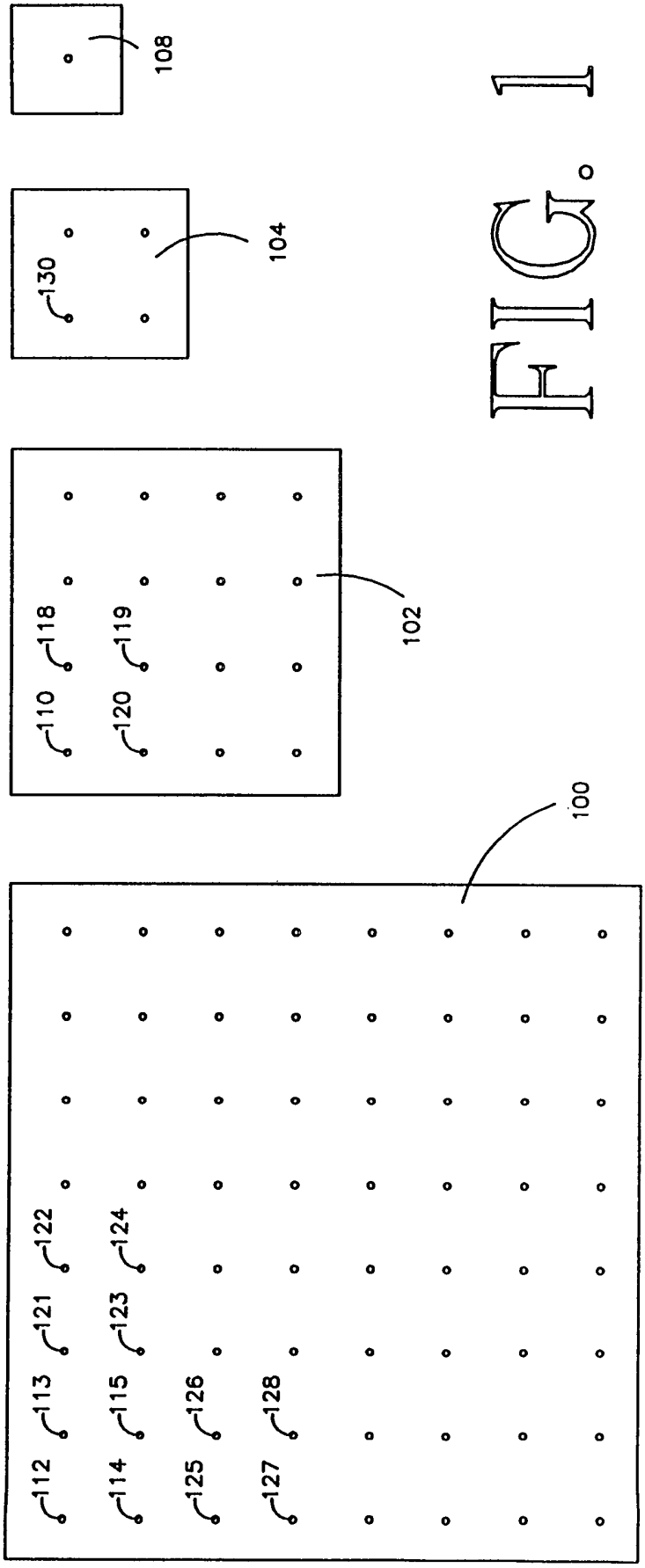
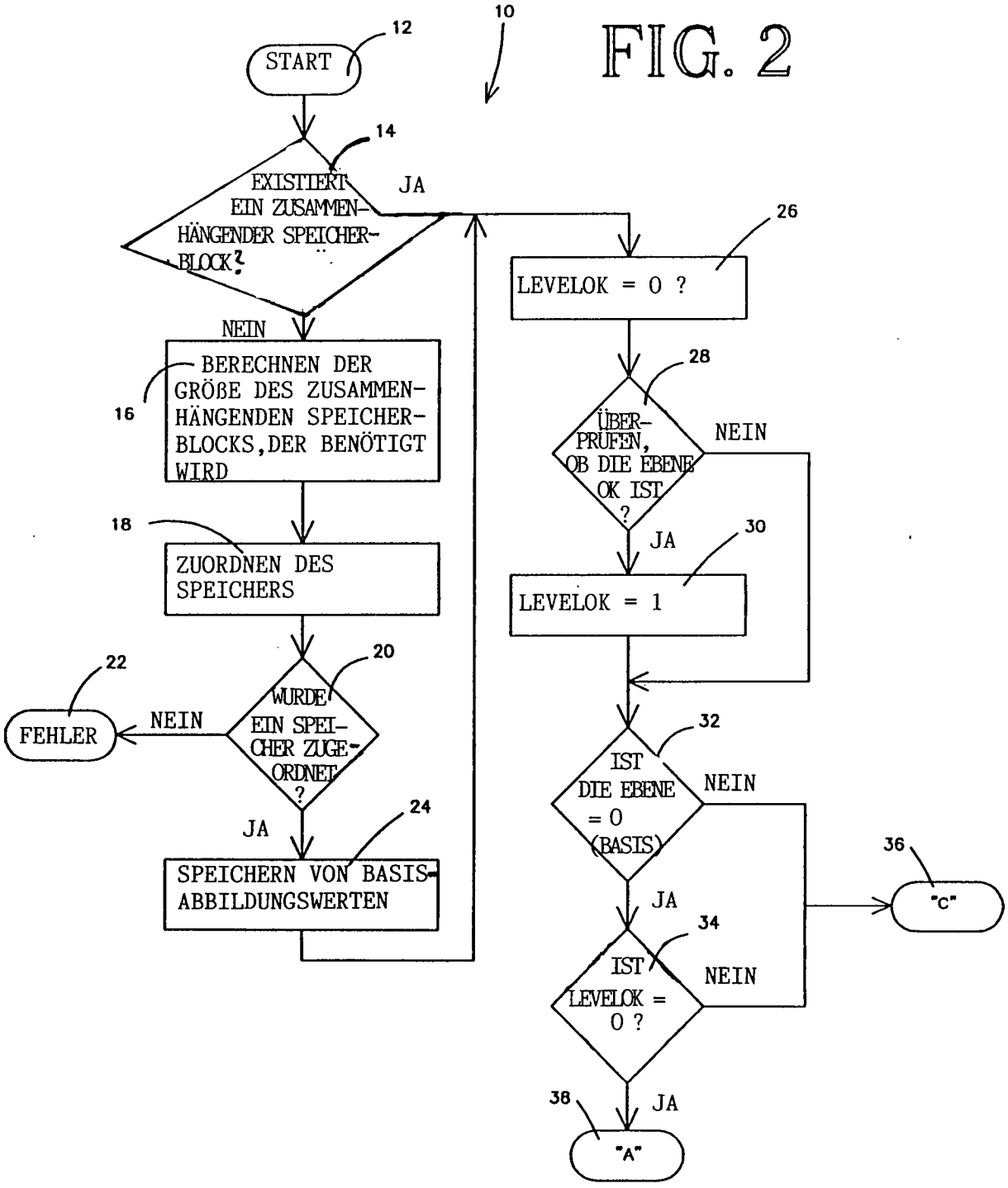


FIG 1

FIG. 2


10
12
START
14
EXISTIERT EIN ZUSAMMEN-HÄNGENDER SPEICHER-BLOCK?
JA
NEIN
16
BERECHNEN DER GRÖßE DES ZUSAMMEN-HÄNGENDEN SPEICHER-BLOCKS, DER BENÖTIGT WIRD
18
ZUORDNEN DES SPEICHERS
20
WURDE EIN SPEI-CHER ZUGE-ORDNET ?
22
FEHLER
NEIN
JA
24
SPEICHERN VON BASIS-ABBILDUNGSWERTEN
26
LEVELOK = 0 ?
28
ÜBER-PRÜFEN, OB DIE EBENE OK IST ?
NEIN
JA
30
LEVELOK = 1
32
IST DIE EBENE = 0 (BASIS)
NEIN
JA
34
IST LEVELOK = 0 ?
NEIN
JA
36
"C"
38
"A"

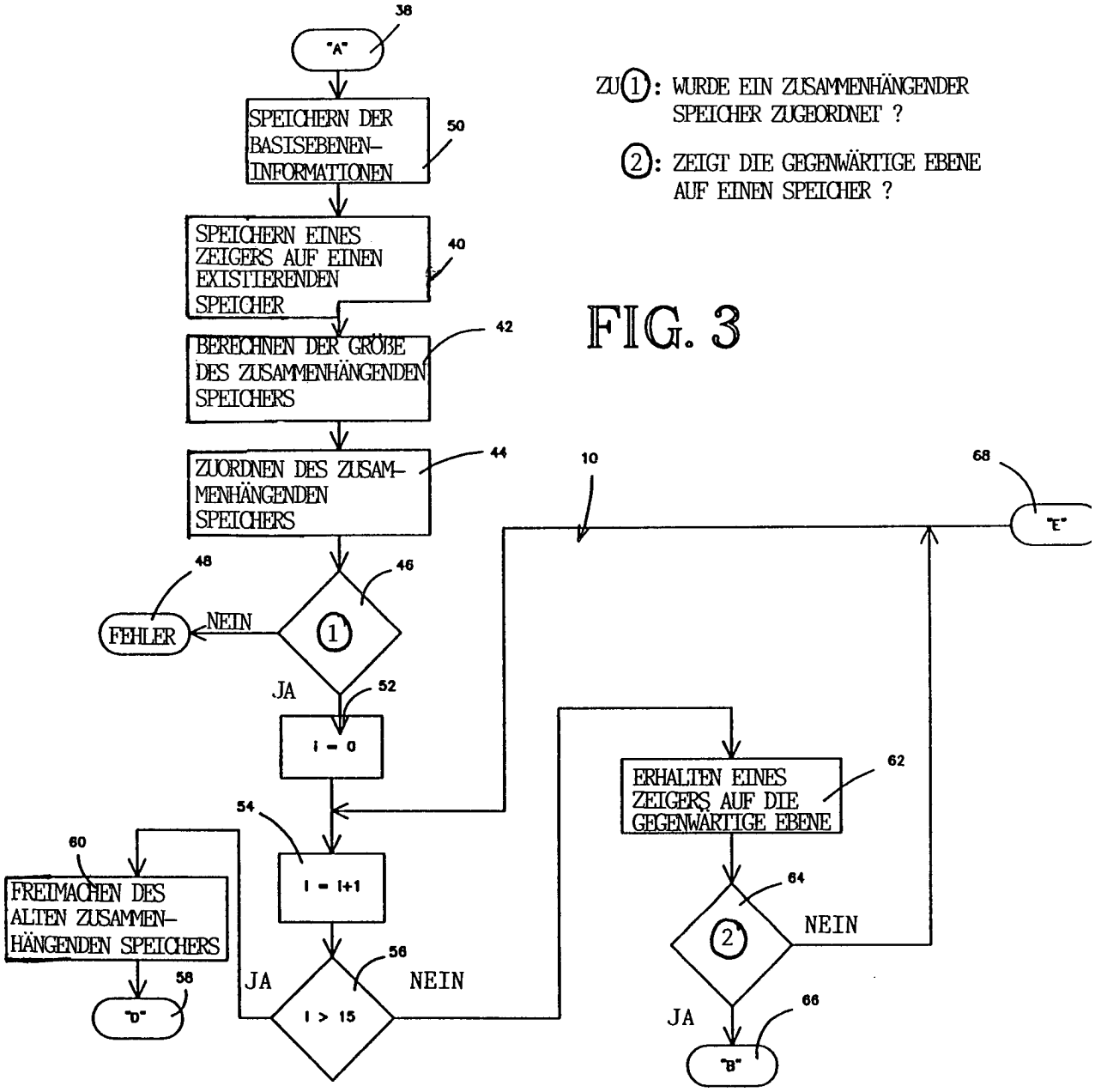

38
"A"
SPEICHERN DER BASISEBENEN-INFORMATIONEN
50
SPEICHERN EINES ZEIGERS AUF EINEN EXISTIERENDEN SPEICHER
40
BERECHNEN DER GRÖßE DES ZUSAMMENHÄNGENDEN SPEICHERS
42
ZUORDNEN DES ZUSAMMENHÄNGENDEN SPEICHERS
44
48
FEHLER
NEIN
46
1
JA
52
i = 0
54
i = i+1
60
FREIMACHEN DES ALTEN ZUSAMMENHÄNGENDEN SPEICHERS
58
"D"
56
JA
NEIN
i > 15
10
68
"E"
ERHALTEN EINES ZEIGERS AUF DIE GEGENWÄRTIGE EBENE
62
64
2
NEIN
JA
66
"B"
ZU 1: WURDE EIN ZUSAMMENHÄNGENDER SPEICHER ZUGEORDNET ?
2: ZEIGT DIE GEGENWÄRTIGE EBENE AUF EINEN SPEICHER ?

FIG. 3

FIG. 4

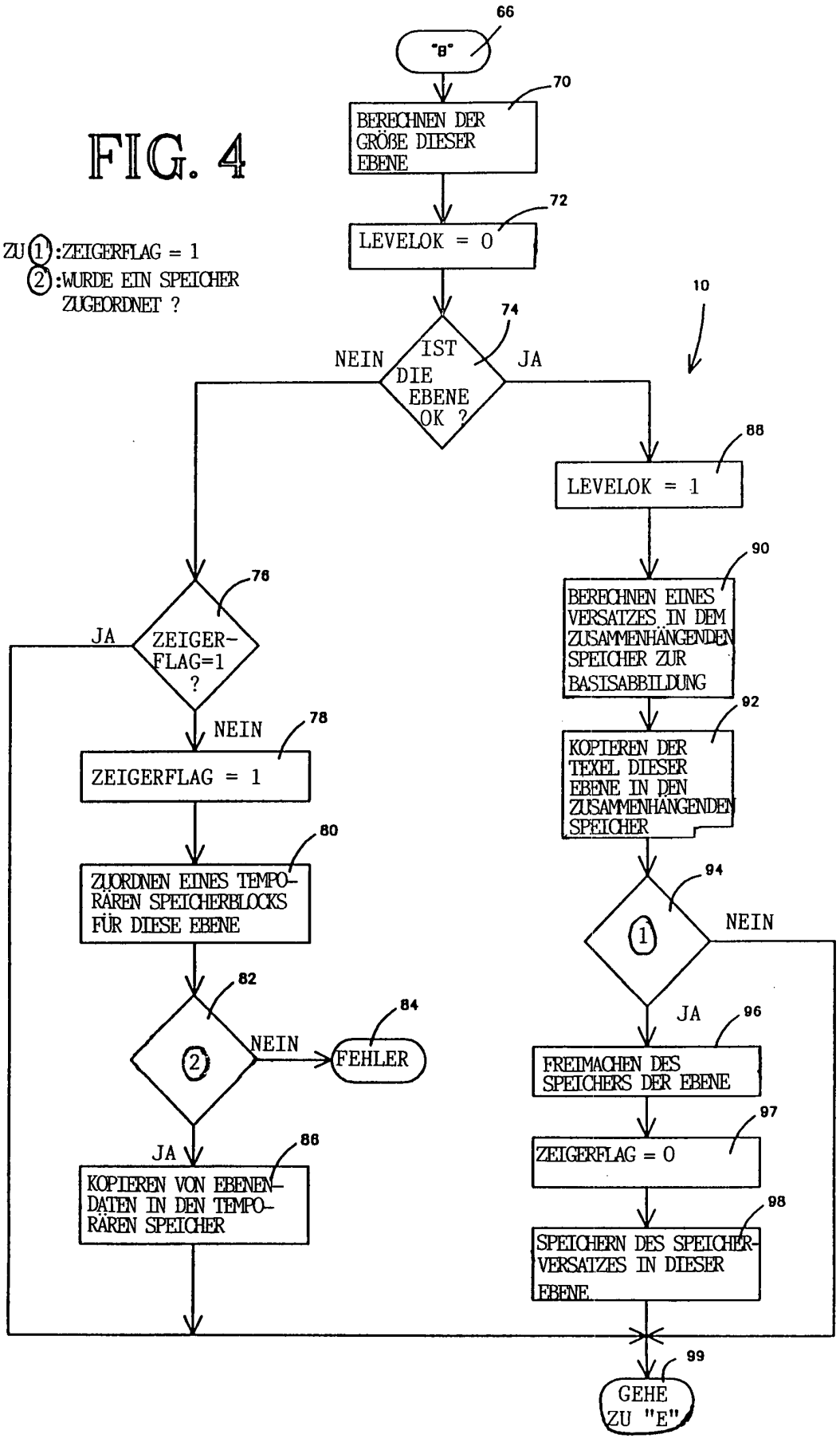
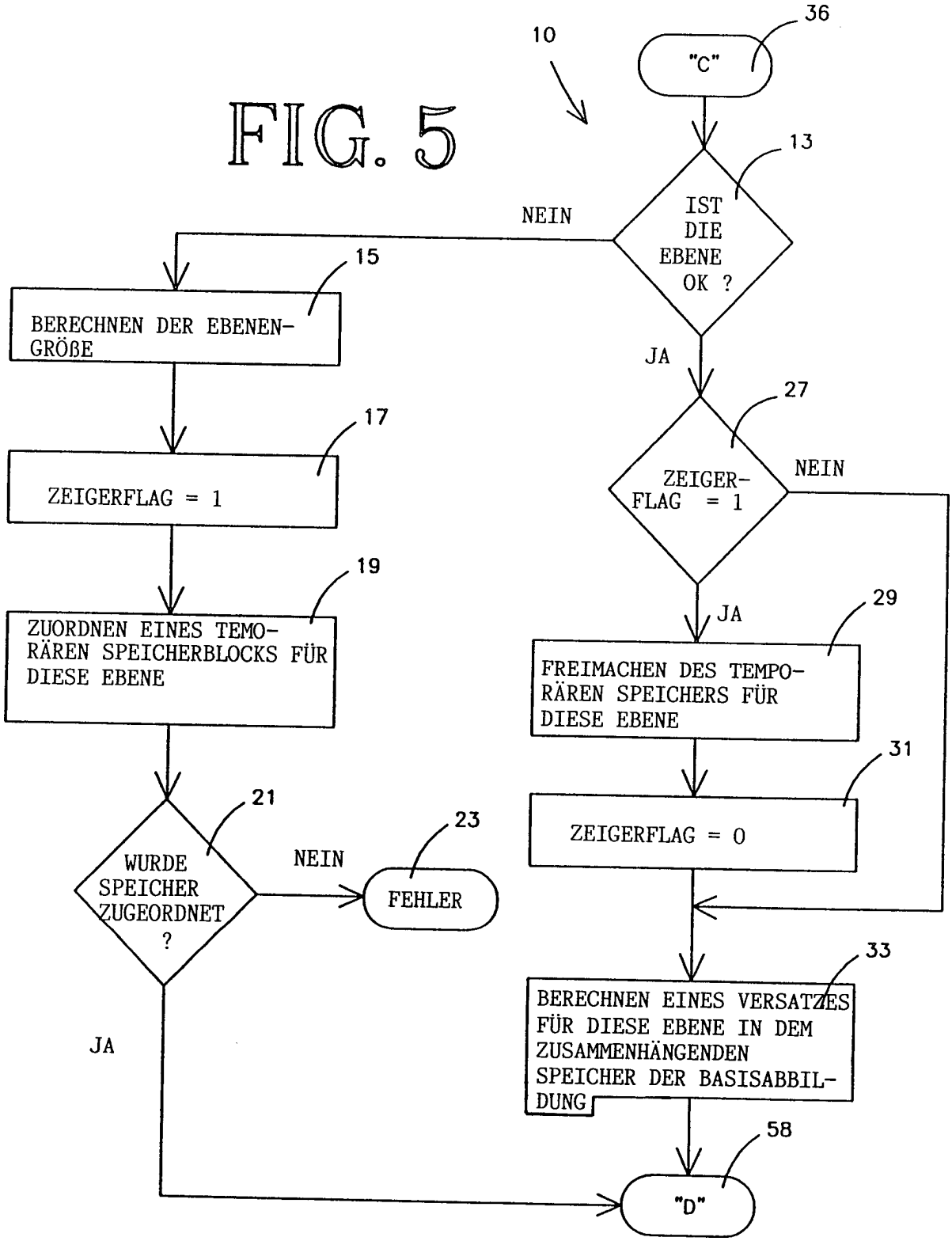

ZU (1):ZEIGERFLAG = 1
(2):WURDE EIN SPEICHER ZUGEORDNET ?
66
"B"
70
BERECHNEN DER GRÖßE DIESER EBENE
72
LEVELOK = 0
10
74
IST DIE EBENE OK ?
NEIN
JA
88
LEVELOK = 1
90
BERECHNEN EINES VERSATZES IN DEM ZUSAMMENHÄNGENDEN SPEICHER ZUR BASISABBILDUNG
92
KOPIEREN DER TEXEL DIESER EBENE IN DEN ZUSAMMENHÄNGENDEN SPEICHER
94
(1)
NEIN
JA
96
FREIMACHEN DES SPEICHERS DER EBENE
97
ZEIGERFLAG = 0
98
SPEICHERN DES SPEICHER-VERSATZES IN DIESER EBENE
76
ZEIGER-FLAG=1 ?
JA
NEIN
78
ZEIGERFLAG = 1
80
ZUORDNEN EINES TEMPORÄREN SPEICHERBLOCKS FÜR DIESE EBENE
82
(2)
NEIN
84
FEHLER
JA
86
KOPIEREN VON EBENEN-DATEN IN DEN TEMPORÄREN SPEICHER
99
GEHE ZU "E"

FIG. 5


10
36
"C"
13
IST DIE EBENE OK ?
NEIN
JA
15
BERECHNEN DER EBENEN-GRÖßE
17
ZEIGERFLAG = 1
19
ZUORDNEN EINES TEMORÄREN SPEICHERBLOCKS FÜR DIESE EBENE
21
WURDE SPEICHER ZUGEORDNET ?
NEIN
23
FEHLER
JA
27
ZEIGER-FLAG = 1
NEIN
JA
29
FREIMACHEN DES TEMPORÄREN SPEICHERS FÜR DIESE EBENE
31
ZEIGERFLAG = 0
33
BERECHNEN EINES VERSATZES FÜR DIESE EBENE IN DEM ZUSAMMENHÄNGENDEN SPEICHER DER BASISABBILDUNG
58
"D"

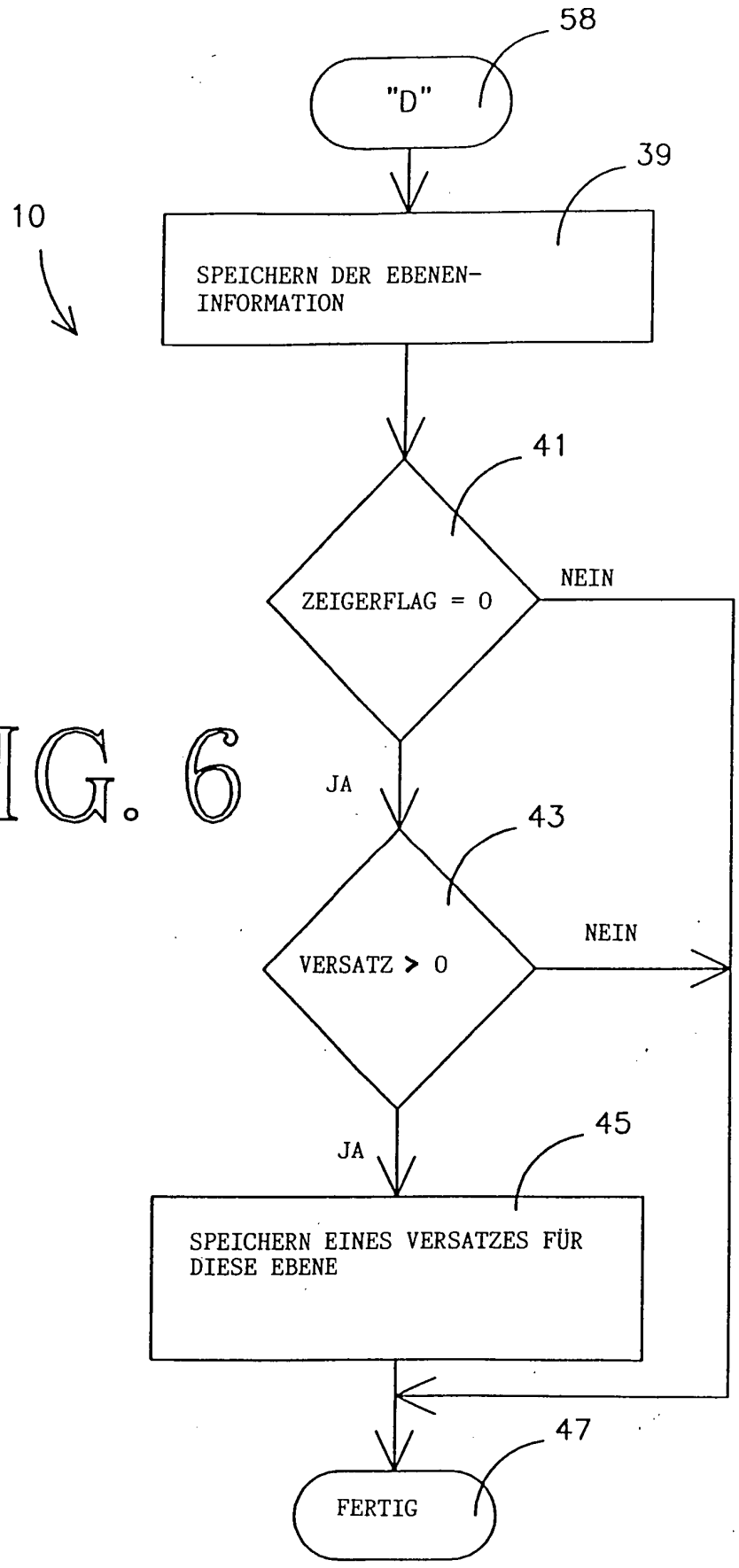

58
"D"
39
10
SPEICHERN DER EBENEN-INFORMATION
41
ZEIGERFLAG = 0
NEIN
JA
43
VERSATZ > 0
NEIN
JA
45
SPEICHERN EINES VERSATZES FÜR DIESE EBENE
47
FERTIG

FIG. 6