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

(10) **DE 699 17 799 T2** 2005.06.23

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 125 251 B1**
(21) Deutsches Aktenzeichen: **699 17 799.5**
(86) PCT-Aktenzeichen: **PCT/GB99/03711**
(96) Europäisches Aktenzeichen: **99 954 175.8**
(87) PCT-Veröffentlichungs-Nr.: **WO 00/28479**
(86) PCT-Anmeldetag: **08.11.1999**
(87) Veröffentlichungstag der PCT-Anmeldung: **18.05.2000**
(97) Erstveröffentlichung durch das EPA: **22.08.2001**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **02.06.2004**
(47) Veröffentlichungstag im Patentblatt: **23.06.2005**

(51) Int Cl.[7]: **G06T 15/20**

(30) Unionspriorität:
9824407 06.11.1998 GB

(84) Benannte Vertragsstaaten:
AT, BE, CH, CY, DE, DK, ES, FI, FR, GR, IE, IT, LI, LU, MC, NL, PT, SE

(73) Patentinhaber:
Imagination Technologies Ltd., Kings Langley, Herfordshire, GB

(72) Erfinder:
Pearce, Simon, Kings Langley, Hertfordshire WD4 8LB, GB

(74) Vertreter:
COHAUSZ & FLORACK, 40211 Düsseldorf

(54) Bezeichnung: **TEXTURIERUNGSSYSTEME ZUR VERWENDUNG IN DREI-DIMENSIONALEN ABBILDUNGSSYSTEMEN**

Beschreibung

Bereich der Erfindung

[0001] Die vorliegende Erfindung betrifft Texturiersysteme für die Verwendung in dreidimensionalen Abbildungssystemen, wie sie beispielsweise in Computerdisplays zum Einsatz kommen. Solche Texturiersysteme werden z.B. in Displays von Computerspielen eingesetzt.

Stand der Technik

[0002] Zu weiteren Informationen über den Hintergrund der vorliegenden Technologie wird auf die folgenden bekannten Dokumente und Arbeiten verwiesen. Auf einige davon wird ggf. nach ihrer Nummer in der folgenden Liste verwiesen:

[1] Catmull, E. „ A Subdivision Algorithm for Computer Display of Curved Surfaces", Ph.D. Thesis, report UTEC-Csc-74–133, Computer Sciences Department, University of Utah, Salt Lake City, UT, Dezember 1974.
[2] Blinn, J.F. and M.E. Newell, "Texture and Reflection in Computer Generated Images", CACM, 19(10), Oktober 1976, 542–547.
[3] Wolberg. G., "Digital Image Warping", IEEE Computer Society Press, Los Alamitos, CA, 1990.
[4] Williams, L., "Pyramidal Parametrics", SiGGRAPH 83, Seiten 1–11.
[5] VideoLogic, "Apocalypse 3Dx data", erhältlich von VideoLogic Limited, Kings Langley, England. [6] Microsoft Corporation, "Texture and Rendering Engine Compression (TREC)", Microsoft Technical Brief, Internet-Adresse:
www.microsoft.com/hwdec/devdes/whntrec.htm
[7] Beers, A.C. Agrawala, M. and Chaddha, N., "Rendering from Compressed Textures", 1996 Computer Graphics Annual Conference, S. 373.
[8] Hakura, Z.S. and Gupta, A. "The design and analysis of a cache architecture for texture mapping", Computer Architecture News, Band. 25, Nr. 2, S. 108–20, Mai 1997.
[9] United States Patent US-A-5,612,747 "Method and apparatus for vector quantization caching in a real time video coder".
[10] UK-Patentanmeldung GB-A-2,297,886 "Texturing and shading of 3-D Images", Anmelder: VideoLogic Limited.
[11] Gray, R.M., "Vector Quantization", IEEE Transactions on Communications, Januar 1980, S. 4–20.
[12] Koegel Buford, J., "Multimedia Systems", Herausgeber Addison-Wesley, 1994.
[13] Foley, F. and van Dam, A., "Computer Graphics Principles and Practice", Herausgeber Addison-Wesley, 1990.
[14] Microsoft, "DirectX 6.0 SDK Documentation", Microsoft Corporation, 1 Microsoft Way, Redmond, USA. Internet-Adresse: www.microsoft.com/directx.

Hintergrund der Erfindung

[0003] Bilder auf Computerbasis werden gewöhnlich aus einer Array von Bildelementen oder Pixeln gebildet. Jede anzuzeigende Fläche kann von den Pixeln in einem Polygon dargestellt werden, gewöhnlich einem Dreieck. Der Fläche wird mit einem als „Texturmapping" bezeichneten Verfahren eine bestimmte Farbe, Textur und/oder Schattierung gegeben. Texturen werden als Arrays von Pixeln gespeichert, die kurz Texel (Texturpixel) genannt werden. Somit beinhaltet „Texturmapping" das Mapping einer zweidimensionalen (2D) Array von Texeln auf eine andere 2D-Array von Pixeln, die eine Fläche in einer dreidimensionalen (3D) Szene repräsentieren. Diese Technik wurde von Catmull [Dok. 1] entwickelt und von Blinn und Newell [Dok. 2] verfeinert. Perspektivisches Texturmapping beinhaltet das Rotieren und Translatieren der Textur-Map, so dass ihre Perspektive auf dem endgültigen Bild korrekt erscheint. Textur-Mapping verbessert den Realismus der Szene, da die Oberfläche eines Objektes ein realistisches Finish erhält. Ein Beispiel hierfür ist das Mapping einer Marmortextur auf die Oberfläche einer Statue, so dass die Statue ein Aussehen erhält, als bestünde sie aus Marmor. Für eine große Szene werden viele verschiedene Textur-Bitmaps benötigt, um alle verschiedenen Texturen darzustellen, die in der Szene vorliegen können.

[0004] Wie soeben bemerkt, wird eine 3D-Szene gewöhnlich mit einer Reihe von Polygonen oder Dreiecken dargestellt. Um ein Polygon mit einer Textur zu füllen, wird jedes Pixel auf seiner Fläche zum Berechnen der Koordinate eines Texels in der Textur-Map benutzt. Das dem in der Textur-Map berechneten Texel am nächsten liegende Texel kann zum Schattieren des schließlich angezeigten Pixels verwendet werden. Dies wird als Punkt-Sampling bezeichnet. Alternativ kann die Qualität des texturierten Bildes mit Hilfe von bilinearer Filte-

rung oder bilinearer Interpolation verbessert werden. Bei bilinearer Filterung wird der Punkt in der Textur-Map, von dem das 2D-Pixel auf der 3D-Fläche abgebildet werden soll, bis auf Subpixel-Genauigkeit berechnet. Mit Hilfe von bilinearer Filterung oder Interpolation werden dann die vier dieser berechneten Position am nächsten liegenden Pixel in der Textur-Map gemischt, um eine genauere Darstellung der Pixelfarbe zu erzielen. Dies ist in der beiliegenden **Fig. 1** illustriert, wo die Texel A, B, C und D gemischt wurden, um einen Texelwert für ein Pixel an Punkt X auf der zweidimensionalen Bildebene zu erzeugen. Dieser Vorgang der bilinearen, d.h. Zweiachsen-Filterung (oder Interpolation) ist in Dok. 3 näher beschrieben.

[0005] Trilineare (Dreiachsen-) Filterung ist derselbe Vorgang über die vier nächstliegenden Pixel auf zwei verschiedenen Mip-Map-Leveln [Dok. 4]. Dies ist in **Fig. 2** der vorliegenden Anmeldung illustriert. Mip-Maps sind Kopien der ursprünglichen Textur-Map, die vorverarbeitet wurden, indem sie gefiltert werden, um sie nacheinander auf die Hälfte der Auflösung zu reduzieren. MIP bedeutet hier MULTUM IN PARVO (viel an einem kleinen Ort). Dies wird so lange wiederholt, bis das resultierende Bild eine Größe von 1 Pixel hat (wobei davon ausgegangen wird, dass die Textur quadratisch und von einer Größe ist, die eine Zweierpotenz ist), so dass es eine hierarchische Serie der Mip-Maps gibt. **Fig. 3** zeigt ein Beispiel für eine Mauerwerktextur mit einer Auflösung von 128 x 128 mit den assoziierten tieferen Mip-Map-Leveln. Eine Mip-Map kann wie eine Pyramide angesehen werden.

[0006] Texturfilterung reduziert das Auftreten von Aliasing-Effekten beim Abtasten von Texturen. Zu weiteren Informationen über Aliasing wird auf Dok. 3 verwiesen.

[0007] Eine dreidimensionale Bilderzeugung ist rechenintensiv. Animierte 3D-Bilder für Spiele und CAD(Computer Aided Design) Anwendungen werden im Hinblick auf die Prozessorleistung immer teurer, da Szenen immer fotorealistischer werden und Bilder immer mehr in Echtzeit reagieren sollen. Es ist eine große Zahl von Floating-Point-Berechnungen erforderlich, um die Geometrie der Polygon-Struktur in der Szene zu bestimmen, und eine große Zahl von Rechenvorgängen wird benötigt, um die Polygone zu füllen und zu schattieren. Es steht dedizierte Hardware zur Verfügung [Dok. 5], die diese Vorgänge um ein Vielfaches effizienter durchführen kann als Software. Zugriffe auf gespeicherte Datenbanken sind ebenfalls ein die Leistung begrenzender Faktor. Lokaler Speicher in dedizierter Hardware kann den Effekt von Speicherzugriffsengpässen reduzieren. Textur-Mapping ist besonders speicherintensiv, wenn ein Filterungs- (d.h. Interpolations-) Vorgang durchgeführt wird, bei dem viele Texturpixel für jedes Pixel gelesen werden, das auf das Display gemappt wird.

[0008] Die Datenmenge einer 2D-Textur-Map wird somit durch Texturkompression reduziert, so dass sie auf einen kleineren Speicherplatz passt. Ein geringerer Speicherbedarf hat niedrigere Systemkosten zur Folge. Die ursprüngliche Textur-Map kann dann durch Dekomprimieren der komprimierten Daten zurückgewonnen werden. In dem Maße, in dem 3D-Szenen immer realistischer werden, werden Textur-Maps immer größer und zahlreicher, so dass Texturkompression immer wichtiger wird. Es wurden bereits mehrere Schemata entwickelt, wie z.B. Texture and Rendering Engine Compression (TREC) von Microsoft [Dok. 6]. Beers [Dok. 7] hat die Technik des Renderns von Bildern von komprimierten Texturen zum ersten Mal erörtert.

[0009] An dieser Stelle sollten die verschiedenen Speichertypen betrachtet und definiert werden, die dem Systemdesigner zur Verfügung stehen. Der Begriff „lokaler Speicher" bezieht sich auf Festkörper-Halbleiterspeicher, die sich in der Nähe des/der Speichersteuerungs-Halbleiterbauelementes oder -schaltung befindet. Der Begriff „interner Speicher" bedeutet Speicher, der sich in dem jeweiligen Halbleiterbauelement befindet, um das es geht. „Externer Speicher" ist beliebiger Speicher außerhalb des Halbleiterbauelementes. Lokaler Speicher kann auf DRAM basieren. DRAM ist eine Abkürzung für Dynamic Random Access Memory (dynamischer Arbeitsspeicher), bei dem es sich um einen Festkörper-Halbleiter handelt. Synchroner DRAM (SDRAM) ermöglicht die Koordination von Datenzugriffen durch ein Taktsignal. SDRAM hat aufgrund seiner Pipeline-Struktur eine höhere Zugriffsbandbreitenfähigkeit als DRAM, ist aber teurer. Lokaler Speicher und interner Speicher können auf DRAM oder SDRAM basieren. Externer Speicher kann eine Festkörper- oder Massenspeicherarray wie z.B. eine Festplatte sein. Halbleiterspeicher ist sehr teuer und macht einen großen Prozentanteil der Gesamtkosten eines Computersystems aus.

[0010] DRAM wird über einen multiplexierten Adressbus adressiert, d.h. die zum Zugreifen auf ein individuelles Datenelement benötigte Adresse wird in zwei Teilen zu dem Speicherbauelement übertragen. Die Kernspeicherarray im DRAM-Bauelement ist eine rechteckige Matrix, wobei ein einzelnes Datenelement adressiert wird, wenn eine Reihensteuerleitung und eine Spaltensteuerleitung gleichzeitig aktiviert werden. Dies erfordert eine separate Reihen- und Spaltenadresse. Wenn sich die Reihenadresse zwischen sequentiellen Zugriffen nicht ändert, dann braucht nur die Spaltenadresse übertragen zu werden. Eine Reihe von Daten in der DRAM-Array wird als Seite bezeichnet. Wenn die Reihenadresse zwischen Zugriffen unverändert bleibt, dann

gelten die Zugriffe als „seitenintern" (in Page). Seiteninterne Zugriffe sind weitaus schneller als solche, die über zwei oder mehr Seiten gehen, und die Designer von Speichersystemen bemühen sich, mehrere Zugriffe seitenintern zu halten. Einige Speicherbauelemente, wie z.B. SDRAM, nutzen mehrere Speicherbänke zum Verbessern der Leistung. Jede Speicherbank kann ihre eigene Seite offen haben, so dass Datenzugriffe auf separate Speicherbereiche ohne Seitenumbruch möglich sind.

[0011] Eine Technik, die zum Verbessern der Speicherleistung eingesetzt wird, ist „Memory Caching", bei dem die Ergebnisse aller externen Speicherzugriffe in einem separaten internen Speicher gespeichert werden. Auf diesen internen Speicher kann weitaus schneller zugegriffen werden als auf externen Speicher. Wenn sich das Ergebnis eines bestimmten Speicherzugriffs bereits im Cache befindet, dann wird anstatt des externen Speichers der Cache gelesen. Dies hat den Effekt, dass Verkehr zum externen Speicher reduziert wird, und somit wird der „Bandbreiten"-Bedarf dieses Speichers reduziert. Der Bandbreitenbedarf eines Speichersystems wird oft direkt auf die Kosten dieses Systems bezogen. In einem System mit fester Bandbreite kann ein höherer Bandbreitenbedarf zu einer Reduzierung der Systemgesamtleistung führen.

[0012] Texturierung ist der leistungsintensivste Prozess der 3D-Abbildung, da sie verlangt, dass alle Texturen aus dem Speicher abgerufen werden. Bei Techniken wie trilineare und bilineare Filterung (Interpolation) müssen für jedes auf das Display projizierte Pixel bis zu acht Texturpixel oder Texel aus dem Speicher abgerufen werden, wie oben beschrieben und in den **Fig. 1** und **Fig. 2** illustriert ist. Texturierung erfordert somit einen Pfad mit sehr hoher Bandbreite in den Speicher. Textur-Caching kann zum Reduzieren des Texturierbandbreitenbedarfs und zum Erhöhen der Systemleistung verwendet werden. Das optimale Leistungsziel ist es, alle notwendigen Texel in einem einzigen Verarbeitungspipeline-Taktzyklus lesen zu können. Es wurden bereits einige Arbeiten im Hinblick auf das Studium der Auswirkungen des Verwendens eines Cache zum Verbessern der Leistung von Texturzugriffen durchgeführt [Dok. 8]. Hakura demonstriert, wie Cache-Speicher äußerst effektiv für ein Textur-Mapping von Graphik eingesetzt werden kann, und kommt zu dem Schluss, dass das effektive Speicher-Bandbreiten-Verhältnis mit bestimmten Caching-Strategien um einen Faktor von drei bis fünfzehn reduziert werden kann.

[0013] Wie zuvor angedeutet, wird mit Hilfe von Textur-Mapping die Qualität von 3D-Bildern durch Mappen von viel Detail auf eine Fläche eines 3D-Objekts verbessert. Dadurch soll die Geometrie des Objekts nicht verkompliziert werden.

[0014] Textur-Mapping erzeugt jedoch eine Vielfalt von visuellen Artefakten einschließlich Aliasing [Dok. 13]. Mit Hilfe von bilinearer Filterung [Dok. 3] wird die Qualität des resultierenden Bildes zwar verbessert, aber es bleiben viele Artefakte, die mit bilinearer Filterung gelöst werden können, einschließlich Tiefen-Aliasing. Tiefen-Aliasing ist die Folge davon, dass die Textur mit zunehmender Entfernung eines Objekts vom Blickpunkt immer stärker komprimiert wird. Diese Form von Aliasing kann mit Hilfe von Mip-Maps [Dok. 4] gelöst werden, aber es besteht weiterhin ein Problem, das mit Mip-Banding bezeichnet wird. Mip-Banding tritt in der Übergangsperiode zwischen Mip-Maps auf, wenn sich die Textur von einem Detail-Level zu einem anderen ändert. Dies kann z.B. auf einer Straße vorkommen, die im Vordergrund zu sehen ist und irgendwo in der Ferne verschwindet.

[0015] Entlang der Straße werden aufeinander folgende Mip-Maps verwendet, und der Übergang von einem Mip-Map zum nächsten kann sichtbar sein. Dieses Problem kann mit der Anwendung von trilinearer Filterung [Dok. 4] gelöst werden, die das Detail-Level zwischen Mip-Maps interpoliert, wie oben beschrieben wurde.

[0016] Die beste Form der trilinearen Filterung ist die, die auf einer Pro-Pixel-Basis durchgeführt wird. Dies erfordert acht Textur-Pixel (Texel) zur Erzeugung des endgültigen Bildschirmpixels. Da sich diese Texel irgendwo im Speicher befinden können, werden häufig acht separate Speicherlesevorgänge benötigt. Zwischen zwei Mip-Levels erfolgt eine trilineare Filterung, und daher kommt es zu vier Speicherlesevorgängen von einem Mip-Map-Ort und zu vier von einem anderen. Texturen werden gewöhnlich in lokalem Speicher gespeichert, aber Systemspeicher-Texturierung wird immer beliebter. Diese Speicher haben eine finite Bandbreite und werden sehr häufig als Speicherressource für viele verschiedene Anwendungen benötigt. Setup-Parameter, Tiefeninformationen und Anzeigeinformationen werden gewöhnlich in lokalem Speicher gespeichert, und Systemanwendungen werden gewöhnlich vom Systemspeicher aus laufen gelassen. Acht individuelle Speicherlesevorgänge pro Pixel liegen gewöhnlich jenseits der Fähigkeiten vieler Speichersysteme. Und dazu noch ein Seitenwechsel zwischen Mip-Maps, ergibt häufig eine unzureichende 3D-Leistung.

[0017] Die für ein trilineares Texturzugriffssystem benötigten Speicherbandbreiten sind von der Zahl der Speicherzugriffe, die für jeden Texturfiltervorgang notwendig sind, sowie von der Pixelgesamtleistung abhängig, die

von der Anwendung gefordert wird. Gleichung 1 zeigt, wie die Texturbandbreite bestimmt werden kann. Die Gleichung zeigt auch, wie die Bandbreite von Seitenumbrüchen ebenfalls berücksichtigt werden muss.

$$Bandwidth_{texture} = ((Accesss_{pixel} \times Width_{memory}) + (Accesses_{page\_break} \times Width_{memory})) \times Throughput_{pixel}$$ Gleichung (1)

[0018] Dabei gilt:
$Bandwidth_{texture}$ ist die in Byte/s gemessene Texturbandbreite, die der Speicher benötigt. Dies ist nicht die Speicherbandbreite, die der Speicher liefern kann.

[0019] $Accesses_{pixel}$ ist die durchschnittliche Anzahl von Speicherzugriffen pro Pixel. Nicht alle benötigten Texel können in einem Zugriff gelesen werden, nicht einmal mit der richtigen Datenbreite.

[0020] $Accesses_{page\_break}$ ist die durchschnittliche Anzahl von Speicherzugriffsslots, die pro Pixel auf Seitenumbrüche verloren gehen. Ein einzelner Seitenumbruch mit SDRAM erfordert wenigstens 8 Zugriffsslots.

[0021] $Width_{memory}$ ist die in Bytes gemessene Breite des Speicherdatenbusses. Sie muss wenigstens 8 Byte (64 Bit) betragen, um sicherzustellen, dass vier Texel in einem Taktzyklus gelesen werden können.

[0022] $Throughput_{pixel}$ ist der in Pixeln/s gemessene Pixeldurchsatz, der von der Anwendung benötigt wird. Für die meisten modernen Anwendungen sind dies etwa 100 Mpixel/s.

[0023] Die durchschnittliche Zahl der Zugriffe pro Pixel ist die Anzahl von separaten Speicherzugriffen, die zum Abrufen aller für den Filtervorgang benötigten Daten erforderlich sind. Unter Verwendung eines 64-Bit-Speicherbusses wird ein maximaler Durchsatz erzielt, wenn vier 16-Bit-Texel benötigt werden, und sie befinden sich im selben Datenwort. Leider ist dies nicht immer der Fall, und sehr häufig befinden sich die Texturdaten in zwei oder vier separaten Wörtern. Gleichung 2 zeigt, wie $Accesses_{pixel}$ gefunden werden können, unter Berücksichtigung der variierenden Anzahl von Zugriffen für einen einzelnen Texturvorgang.

$$Accesses_{texture} = \frac{((Percentage_{single} x1) + (Percentage_{double} x2) + (Percentage_{quadruple} x4) x Mipmaps}{Factor_{compression}}$$ Gleichung (2)

[0024] Dabei gilt:
$Percentage_{single}$ ist der Prozentanteil der einzelnen Mip-Map-Zugriffe, die in einem Speicherzugriff abgerufen werden können.

[0025] $Percentage_{double}$ ist der Prozentanteil der einzelnen Mip-Map-Zugriffe, die in zwei Speicherzugriffen abgerufen werden können.

[0026] $Percentage_{quadruple}$ ist der Prozentanteil der einzelnen Mip-Map-Zugriffe, die in vier Speicherzugriffen abgerufen werden können.

[0027] Mipmaps ist die Anzahl von Mip-Maps, die an dem Filtervorgang beteiligt sind.

[0028] $Factor_{compresion}$ ist der Kompressionsfaktor, wenn die Textur komprimiert wird.

[0029] Die durchschnittliche Anzahl der Zugriffe wird unter Berücksichtigung der Wahrscheinlichkeit errechnet, dass alle Daten in einem, in zwei oder in vier Zugriffen abgerufen werden. Die Gleichung berücksichtigt auch die Anzahl der Mip-Maps, die im Filtervorgang benötigt werden, trilineare Filterung braucht zwei Zugriffe, aber bilineare nur einen. Die Gleichung zeigt auch, wie die durchschnittliche Anzahl der Zugriffe mit Kompression linear reduziert wird. Je weniger Daten abzurufen sind, desto weniger Datenzugriffe sind offensichtlich nötig, und desto wahrscheinlicher wird es, dass sich alle Daten im selben Datenwort befinden.

[0030] Gleichung 1 zeigt somit, dass der physikalische Speicherbandbreitenbedarf selbst mit der Benutzung von Textur-Caches immer noch außerhalb des Rahmens eines rentablen Speichersystems liegt. Aus diesem Grund wird Texturkompression verwendet, nicht nur zum Reduzieren der physikalischen Größe der gespeicherten Textur, mit der damit einher gehenden Reduzierung der Speicherkosten, sondern auch zum Reduzieren des Volumens an Texturdaten, die von diesem Speicher übertragen werden.

[0031] Gleichung 2 zeigt, wie der Kompressionsfaktor die Anzahl der Speicherzugriffe beeinflusst. Dedizierte Hardware hoher Leistung kann zum Dekomprimieren der Texturen in Echtzeit verwendet werden, nachdem diese aus dem Speicher gelesen wurden. Es wurden zahlreiche Texturkompressionstechniken versucht, wie z.B: Vektorquantisierung (VQ) [Dok. 11]; Farb-Lookup-Tabellen (CLUT) oder Palletisation [Dok. 12]; Discrete Cosine Transformation (DCT) [Dok. 12]; sowie mehrere proprietäre Techniken [Dok. 6 und 14]. Mit jeder Technik sind jedoch Probleme assoziiert: VQ und Palletisation erfordern zwei Speicherlesevorgänge oder große interne Daten-Caches, und die Qualität kann begrenzt sein; DCT erfordert eine große Menge an Dekompressionslogik, und dies kann bei einem beschränkten Siliciumetat undurchführbar sein; und viele proprietäre Techniken bieten begrenzte Kompressionsverhältnisse und Qualität. Wie Gleichung 1 demonstriert, können diese Techniken den Bandbreitenbedarf von trilinearer Filterung nur zum Teil lösen.

[0032] Speicherzugriffsströme, die ständig zwischen verschiedenen Speicherbänken wechseln, können einen großen Einfluss auf die Leistung haben. Gleichung 1 zeigt, welchen Einfluss dominante Seitenumbrüche auf die Leistung von Texturfilterung insgesamt haben. Für trilineare Filterung können Seitenumbrüche besonders problematisch sein, wo eine Reihe von Mip-Maps häufig mehr als eine Speicherseite überspannen.

[0033] 3D-Abbildungstechniken verlangen häufig ein so hohes Leistungsniveau, dass nur dedizierte Hardware-Lösungen in Frage kommen. Dies erfordert häufig die Entwicklung eines speziellen Siliciumchips. Der Chip führt nicht nur Textur-Mapping aus, sondern wird auch häufig für die Durchführung sämtlicher Geometrieverarbeitung, Polygon-Setup, Bildprojektion, Beleuchtungsberechnungen, Fogging-Berechnungen, Beseitigung verborgener Flächen und Display-Steuerung eingesetzt. Es ist daher wesentlich, dass jede Stufe in der Erzeugung eines 3D-Bildes möglichst klein gemacht wird, damit alle Prozesse auf denselben Siliciumchip passen. Ein trilinearer Filtervorgang braucht nicht nur eine große Speicherbandbreite, sondern kann auch nur in einer großen Logikmenge und somit Siliciumfläche implementiert werden. Es ist wichtig, dass eine optimale Lösung gefunden wird, die die benötigte Logik und somit die Kosten auf ein Minimum begrenzt.

[0034] Die WO-A-974 1534 offenbart ein Verfahren und ein System zum Texturieren für die Verwendung in der dreidimensionalen Abbildung. Dieses System hat einen Speicher zum Speichern von Mip-Map-Daten, einen Eingang zum Empfangen von Eingabedaten, die den Typ von Mip-Map-Daten anzeigen, ein Control zum Abrufen der Mip-Map-Daten aus dem Speicher gemäß den Eingaben, einen Cache zum Speichern von Teilen von Mip-Map-Daten, die aus dem Speicher abgerufen wurden, und einen trilinearen Interpolator zum Interpolieren eines Ausgangstexels von im Cache gespeicherten Mip-Map-Daten.

[0035] Aus dem Obigen ist ersichtlich, dass die Anforderungen im Hinblick auf Speicherkapazität, Geschwindigkeit und Konstruktionsfreundlichkeit des Chips erheblich sind und in der 3D-Abbildung umfassend beansprucht werden, besonders bei Textur-Mapping. Selbst bei Verwendung aller verfügbaren Techniken zum Erfüllen der Anforderungen lassen sich die Beschränkungen trotzdem nur sehr schwer erfüllen, wenn Echtzeit-Bilder hoher Qualität erzielt werden sollen.

Zusammenfassung der Erfindung

[0036] Die Erfindung ist in ihren verschiedenen Aspekten in den unabhängigen Ansprüchen unten definiert, auf die nunmehr Bezug genommen werden sollte. Vorteilhafte Merkmale sind in den beiliegenden Ansprüchen dargelegt.

[0037] Bevorzugte Ausgestaltungen der Erfindung werden nachfolgend mit Bezug auf die Zeichnungen beschrieben. In diesen Ausgestaltungen wird die Effizienz von trilinearer Texturfilterung durch die Erzeugung des Lower-Level-Textur-Mip-Maps nebenbei von dem Upper-Level-Mip-Map sowie mit Hilfe von Textur-Caching und Texturdekompression als Mittel zum Erfüllen der hohen Speicherbandbreitenanforderungen verbessert. Da die Notwendigkeit zum Lesen des Lower-Level-Mip-Maps in einem separaten Speicherzugriff entfällt, entfallen Seitenumbruchprobleme und die Leistung wird verbessert.

[0038] Insbesondere umfassen die bevorzugten Texturiersysteme einen Speicher zum Speichern von Mip-Map-Daten für die Verwendung beim Texturieren eines Bildes, wobei die Mip-Map-Daten eine hierarchische Serie von Mip-Maps von verschiedenen Levels von abnehmender Auflösung umfassen. Daten werden an einem Eingang empfangen, der den Typ von benötigten Mip-Map-Daten und den Level der Mip-Map (s) anzeigt, von der/denen die Daten genommen werden. Ein Controller ruft die gemäß den eingegebenen Daten benötigten Mip-Map-Daten aus dem Speicher ab, und diese Daten werden in einem Cache gespeichert. Ein Lower-Level-Mip-Map-Generator erzeugt Teile der Mip-Map, die in der hierarchischen Serie unmittelbar unter der Mip-Map sind, von der Teile im Cache gehalten werden. Ein trilinearer Interpolator empfängt die Ca-

che-Mip-Map-Daten von einem Mip-Map-Level und die Lower-Level-Mip-Map-Generatordaten vom nächsttieferen Mip-Map-Level, und interpoliert ein Ausgangstexel von Eingangstexeln von den beiden Mip-Map-Leveln.

[0039] Einige Datenkompressionsschemata sind gut für eine Erzeugung von tieferen Mip-Maps nebenbei, indem sie die benötigte Menge an dekomprimierten Daten erzeugen und indem sie den zum Erzeugen der Lower-Level-Mip-Map benötigten Filteralgorithmus unterstützen. Diese Techniken können zum Verbessern der Leistung und der Qualität eines gerenderten 3D-Bildes verwendet werden, während das Hardware-Overhead minimal gehalten wird.

[0040] In einer bevorzugten Ausgestaltung werden die Texturdaten durch arbiträre komprimierte Codes dargestellt, in denen gewählte komprimierte Code-Werte Hauptfarben und andere komprimierte Code-Werte Zwischenfarben definieren, die durch gewählte gewichtete Mittelwerte von Hauptfarben gebildet werden, wobei die entsprechenden Code-Werte ebenso gewichtete Mittelwerte der Code-Werte der gewählten Hauptfarben sind. Der Lower-Level-Mip-Map-Generator interpoliert ein Ausgangstexel aus einer Mehrzahl von Eingangstexeln durch Bearbeiten der komprimierten Code-Werte.

Kurze Beschreibung der Zeichnungen

[0041] Die Erfindung wird nunmehr beispielhaft mit Bezug auf die Zeichnungen näher beschrieben. Dabei zeigt:

[0042] Fig. 1 ein Diagramm, das bilineare Filterung der Textur-Map in einem Abbildungssystem illustriert;

[0043] Fig. 2 ein ähnliches Diagramm, das trilineare Filterung der Textur-Map in einem Abbildungssystem illustriert;

[0044] Fig. 3 eine Serie von Mip-Maps;

[0045] Fig. 4 ein schematisches Blockdiagramm eines Teils eines Abbildungssystems gemäß der vorliegenden Erfindung;

[0046] Fig. 5 ein Diagramm, das den Betrieb der Ausgestaltung von Fig. 4 illustriert;

[0047] Fig. 6 die Speicherung von Texturwörtern in einem ungünstigsten Szenario;

[0048] Fig. 7 ein schematisches Blockdiagramm ähnlich Fig. 4 einer zweiten Ausgestaltung der Erfindung unter paralleler Verwendung von vier Caches und vier Dekompressionseinheiten;

[0049] Fig. 8 auf ähnliche Weise wie Fig. 6 die Speicherung von Texturwörtern in einem günstigsten Szenario;

[0050] Fig. 9 ein Blockdiagramm einer Modifikation der Ausgestaltung von Fig. 7, wobei die Dekompressionseinheiten dynamisch neu zugewiesen werden;

[0051] Fig. 10 ein Blockdiagramm einer weiteren Modifikation der Schaltung von Fig. 7; und

[0052] Fig. 11 und Fig. 12 eine Illustration, wie Interpolation direkt auf komprimierte Textur-Codes angewendet werden kann.

[0053] Ausführliche Beschreibung der bevorzugten Ausgestaltungen Fig. 4 zeigt eine spezifische Ausgestaltung eines Texturiersystems gemäß der Erfindung. Das Texturiersystem **20** umfasst ein Speichersystem **22**, das Teil des vom Abbildungssystem insgesamt verwendeten Speichers ist und das auf eine bekannte Weise von einem Speichercontroller **24** gesteuert wird. Der Speicher **22** speichert unter anderem eine komprimierte Textur-Map, und der Ausgang des Speichers durch den Speichercontroller **24** besteht aus komprimierten Texturdaten von dem angeforderten Teil der Textur-Map. Die Textur definierende Parameter, die für eine beliebige Fläche benötigt werden, werden an einem Eingang **26** empfangen und an einen Texturadressgenerator **28** zum Adressieren desjenigen Teils des Speichers **22** angelegt, an dem sich die gewünschte Textur befindet. Die komprimierte Textur wird vom Speichercontroller **24** aus dem Speicher abgerufen und in einem Textur-Cache **30** gespeichert. Die abgerufene Textur entspricht einem besonderen Texturtyp, der den anzuzeigenden Flächentyp darstellt. Dies kann beispielsweise ein Teil einer Steinmauer oder die Oberfläche einer Straße sein.

Diese Flächentextur wird wie oben beschrieben in einer Mip-Map-Form gehalten, in einer komprimierten Mip-Map-Form, und die verwendete Mip-Map ist die nächste vor der Fläche, wobei der Abstand zwischen der Fläche und dem Beobachter berücksichtigt wird.

[0054] Der Ausgang vom Textur-Cache **30** wird an eine Dekompressionseinheit **32** angelegt, die die Texturwerte zum Bereitstellen von dekomprimierten Texeln dekomprimiert oder decodiert. Die dekomprimierten Texel werden dann sowohl an einen trilinearen Interpolator **34** als auch an einen Generator **36** zum Erzeugen des relevanten Teils der Mip-Map angelegt, die dem im Textur-Cache **30** enthaltenen am nächsten liegt. Der Ausgang des Generators **36** wird ebenfalls an den trilinearen Interpolator **34** angelegt. Somit empfängt der Interpolator die Texel von der Mip-Map vor und hinter dem betrachteten Flächenpunkt, so dass eine trilineare Interpolation, wie in Fig. 2 illustriert, ausgeführt werden kann. Der trilineare Interpolator erzeugt ein trilinear texturiertes Pixel an einem Ausgang **38**.

[0055] Beim Betrieb speist der Texturadressgenerator **28** daher die Adressen der acht trilinearen Texel in den Textur-Cache **30**, der dann die assoziierten komprimierten Texturdaten aus dem Speicher **22** abruft. Die Dekompressionseinheit (DU) **32** dekomprimiert dann 16 Upper-Mip-Map-Texel von den komprimierten Daten. Es werden vier als die angeforderten Upper-Level-Texel ausgewählt, und alle 16 werden vom Lower-Mip-Map-Generator **36** zum Erzeugen der Lower-Level-Mip-Map-Textur verwendet. Der Grund ist, dass vier Upper-Level-Texel zum Erzeugen jedes Lower-Level-Texels verwendet werden, wie nachfolgend beschrieben wird. Diese Upper-Level- und Lower-Level-Mip-Maps werden dann vom trilinearen Interpolator **34** zum Erzeugen des einzigen trilinearen Pixels verwendet. Es wird zwar eine große Zahl von 16 Texeln benötigt, damit diese Technik funktionieren kann, was unkomprimiert eine große Speicherbandbreite bedeuten würde, aber im komprimierten Zustand bedeutet dies eine erhebliche Speicherbandbreitenverbesserung mit minimalen Seitenumbrüchen.

[0056] Die Lower-Level-Mip-Map wird durch vier parallele Filter oder Interpolatoren **40** erzeugt, die die 16 Upper-Level-Texel nehmen und die vier unteren Levels in einem Taktzyklus erzeugen können. Fig. 5 illustriert diesen Vorgang. Von den 16 ursprünglichen Texeln werden vier zum Erzeugen der vier Upper-Level-Texel decodiert, die direkt für trilineare Filterung verwendet werden. Diese 16 werden auch in vier Quadranten unterteilt, jeweils mit vier Pixeln, die dann von vier digitalen Tiefpassfiltern gefiltert werden, um die vier Low-Level-Mip-Map-Texel zu erzeugen, die für die trilineare Filterung benötigt werden. Dieser Filteralgorithmus ist in Gleichung 3 dargestellt:

Lower-level color value0 = (upper0 + upper1 + upper4 + upper5)/4

Lower-level color value1 = (upper2 + upper3 + upper6 + upper7)/4

Lower-level color value2 = (upper8 + upper9 + upper12 + upper13)/4

Lower-level color value3 = (upper10 + upper11 + upper14 + upper15)/4 Gleichung (3)

[0057] Durch Anwenden von tatsächlichen Werten auf Gleichung 1 und Gleichung 2 können die Kosten für eine trilineare Filterung ermittelt werden. Es wurde ermittelt, dass in einem konventionellen System vier Zugriffe auf 64-Bit-Daten für jedes trilineare Pixel ohne Kompression benötigt werden, bei durchschnittlich zwei Seitenumbrüchen pro Pixel. Dies ergibt einen Gesamtbandbreitenbedarf von 16 Gbyte/Sek. für ein System, das 100 Mpixel pro Sekunde erzeugen muss. Eine solche Leistung ist unakzeptabel, aber selbst wenn die Texturdaten gecacht und komprimiert werden, ist die benötigte Bandbreite nicht viel besser (13,2 Gbyte/s). Wenn der doppelte Seitenumbruch jedoch durch Erzeugen der Lower-Mip-Map „nebenbei" im Generator **36** von Fig. 4 entfernt wird, wird gemäß der vorliegenden Erfindung die Bandbreite auf ein akzeptables Niveau reduziert. Ein gecachter und komprimierter Texturbandbreitenbedarf für dieses System liegt bei nur etwa 200 Mbyte/s. Ein mit 100 MHz laufendes 64-Bit-Speichersystem hat eine Spitzenbandbreitenfähigkeit von 800 Mbyte/s, so dass die benötigten 200 Mbyte/s gut im Rahmen ihrer Kapazität liegen. In der Tat kann das die Erfindung ausgestaltende Lower-Level-Mip-Map-Konstruktionssystem Reservebandbreite zurückhalten, die für andere Zwecke zur Verfügung steht.

[0058] Zusammenfassend kann man daher sehen, dass das Texturiersystem von Fig. 4, das für die Verwendung als Teil des dreidimensionalen Abbildungssystems ausgelegt ist, den Speicher **22** zum Speichern von Mip-Map-Daten für die Verwendung bei der Texturierung eines Bildes beinhaltet, wobei die Mip-Map-Daten eine hierarchische Serie von Mip-Maps von verschiedenen Niveaus von abnehmender Auflösung umfassen. Der Eingang **26** empfängt Eingabedaten, die den benötigten Mip-Map-Datentyp sowie den Level der Mip-Map

(s) angibt, von der/denen Daten genommen werden sollen. Der Controller **24** ist mit dem Eingang und dem Speicher zum Abrufen der gemäß den Eingabedaten benötigten Mip-Map-Daten aus dem Speicher gekoppelt. Der Cache **30** ist mit dem Controller gekoppelt, um Teile von aus dem Speicher abgerufenen Mip-Map-Daten zu speichern, die sich auf ein gewähltes Mip-Map-Level beziehen. Der trilineare Interpolator **34** ist mit dem Cache gekoppelt, um Mip-Map-Daten von einem Mip-Map-Level, nämlich dem oberen Level der beiden Levels, von denen Daten benötigt werden, zu empfangen, und ein Ausgangstexel von Eingangstexeln von zwei Mip-Map-Leveln zu interpolieren. Gemäß der Erfindung beinhaltet das Texturiersystem auch den Lower-Level-Mip-Map-Generator **36**, der zwischen dem Cache und dem trilinearen Interpolator geschaltet ist, und erzeugt in Echtzeit Teile der Mip-Map unmittelbar unter (in der hierarchischen Serie) der Mip-Map, von der Teile im Cache gespeichert sind.

[0059] Die in Fig. 4 illustrierte Ausgestaltung der Erfindung zeigt, wie ein Cache **30** und eine Dekompressionseinheit (DU) **32** zum Zurückgeben aller Daten benötigt werden, die zum Konstruieren eines kompletten trilinearen interpolierten Pixels notwendig sind. Ein Problem gibt es jedoch dann, wenn der Filteralgorithmus Daten von zwei oder vier zusätzlichen Datenwörtern benötigt. Fig. 6 zeigt die ungünstigste Situation. Man betrachte die Textur-Bitmap in Fliesen unterteilt, wobei jede Fliese die Texturpixel repräsentiert, die von einem komprimierten Datenwort gebildet werden. In diesem Beispiel enthält das Datenwort **16** Texturpixel. Die dunkler umrandeten Kästen in Fig. 6 bedeuten die Fliesen. In diesem Beispiel erfordert der Filteralgorithmus Texturpixel von vier separaten komprimierten Datenwörtern. Für ein einziges Cache-/Dekompressionssystem erfordert diese ungünstigste Situation vier Cache-Lesevorgänge und somit ein Minimum an vier Taktzyklen. Dieses Problem liegt, in einem geringeren Ausmaß, auch dann vor, wenn zwei separate Datenwörter benötigt werden.

[0060] Eine Konstruktion mit vier Caches und vier DUs kann alle benötigten Daten in einer Taktperiode bereitstellen, unabhängig davon, wo sich die komprimierten Daten befinden. Fig. 7 zeigt ein Blockdiagramm einer zweiten Ausgestaltung der Erfindung, die eine solche ‚Quad Cache'-Anordnung umfasst. Das Texturiersystem von Fig. 7 ist dem System von Fig. 4 ähnlich, und es werden nur die Unterschiede ausführlich beschrieben. In der zweiten Ausgestaltung von Fig. 7 sind das Speichersystem **22** und der Eingang **26** wie in Fig. 4. In diesem Fall hat der Texturadressgenerator **28** vier Ausgänge, um die Situation zu bewältigen, bei der die Ausgänge von maximal vier Fliesen benötigt werden. Die vier Ausgänge bieten jeweils Texturadressen 1 bis 4 und diese werden an Textur-Caches 1 bis 4 (mit **60** bezeichnet) angelegt. Diese kommunizieren mit dem Speichercontroller **24** durch einen Arbiter **58**, um die Texel jeweils von den vier Fliesen zu empfangen. Sie werden in den vier Caches gespeichert, die somit vier gecachte komprimierte Texturen ausgeben, nämlich jeweils gecachte komprimierte Texturen 1 bis 4.

[0061] Die Ausgänge der vier Caches **60** werden jeweils an vier Dekompressionseinheiten **62** angelegt, die jeweils der Dekompressionseinheit von Fig. 4 ähnlich sind. Jede Dekompressionseinheit **62** legt ihre dekomprimierten Daten an eine obere und eine untere Decodiereinheit **68**, **70** sowie an einen Low-Level-Mip-Map-Filter **66** an. Die Ausgänge der oberen und der unteren Decodiereinheit **68**, **70** werden jeweils an einen trilinearen Interpolator **64** angelegt.

[0062] Beim Betrieb ist jeder Cache **60** für ein anderes Segment des Speichers auf der Basis der Größe eines Cache-Wortes verantwortlich, wobei die beiden tieferen Bits der Texturadresse jedes Cache-Wort repräsentieren. Auf diese Weise kann garantiert werden, dass nicht von einem einzigen Cache verlangt wird, mehr als ein Datenwort für ein komplexes trilineares Pixel bereitzustellen.

[0063] Ohne die Lower-Level-Mip-Map von der oberen erzeugen zu können, würden acht Caches und DUs benötigt, vier für die obere und vier für die untere Map. Somit spart die Erzeugung von Lower-Level-Mip-Maps nach Bedarf „nebenbei" Logik (Siliciumfläche) und verbessert auch die Leistung. In Verbindung mit Kompression wird auch die Menge an benötigten Daten stark reduziert.

[0064] Man wird verstehen, dass die Lower-Level-Mip-Maps weiterhin im Speicher gespeichert sind. Sie werden für Pixel benötigt, die noch weiter vom Betrachter weg sind. Es hat sich jedoch in der trilinearen Interpolation als effizienter erwiesen, ihre Anwesenheit zu ignorieren und diejenigen Texel, die für die bearbeiteten Pixel benötigt werden, mit den Generatoren **62** zu regenerieren.

[0065] Einer der Hauptvorteile des Erzeugens von Lower-Level-Mip-Maps von komprimierten Upper-Level-Mip-Maps ist die Menge an Logikoptimierung, die stattfinden kann. Weniger Logik bedeutet kleinere Hardware und geringere Kosten. Ein solcher Optimierungsbereich ist in den Caches und Dekompressionseinheiten.

[0066] Die Caches **60** für das Quad-Cache-System werden effektiv speichergemappt, d.h. jeder Cache enthält Daten für eine separate Texturspeicherfliese, wie oben erwähnt. Aber die DUs **62** brauchen nicht unbedingt auch speichergemappt werden. Speichermapping von DUs würde verlangen, dass jede mit einem bestimmten Cache assoziiert ist. Dies würde es auch erfordern, dass jede DU wenigstens vier Upper-Level-Texel sowie vier Lower-Level-Texel in einem Zyklus erzeugen kann, um die günstigste Situation zu berücksichtigen. Die günstigste Situation ist in Fig. 8 illustriert, wo alle Fliesen von einem Datenwort kommen.

[0067] Indem zugelassen wird, dass jede DU dynamisch auf der Basis der benötigten Ressourcen neu zugewiesen wird, können dann mehrere Hardware-Optimierungen stattfinden, um die Größe der benötigten Hardware drastisch zu reduzieren. Wir haben festgestellt, dass nicht mehr als eine Einheit vier parallele Upper-Level-Texel bereitzustellen braucht, nicht mehr als zwei Einheiten zwei parallele Upper-Level-Texel zu erzeugen brauchen und keine mehr als einen Lower-Level-Texel bereitzustellen braucht.

[0068] Fig. 9 zeigt, wie diese Optimierungen in ein modifiziertes Quad-Cache-System passen. Die modifizierte Anordnung von Fig. 9 ist der Anordnung von Fig. 7 ähnlich, und es brauchen nur die Unterschiede beschrieben zu werden. Dies sind der Einschluss eines Dekompressionseinheits- (DU) Zuordners **82**, der zwischen den Ausgängen der vier Caches **60** und den Eingängen der vier Dekompressionseinheiten **62** geschaltet ist, und eines entsprechenden Zuordnungsaufhebers **84**, der die obere und die untere Decodiereinheit **68**, **70** ersetzt und zwischen den Ausgängen der vier DUs **62** und der Filter **66** sowie den Eingängen zum trilinearen Interpolator **64** geschaltet sind. Die DUs **62** sind nicht mehr identisch; die primäre DU kann vier parallele Upper-Level-Texel bereitstellen, die sekundäre DU kann zwei parallele Upper-Level-Texel bereitstellen, und die tertiäre und quartäre DU können jeweils nur ein Upper-Level-Texel bereitstellen. Die DU-Zuordnungseinheit **82** ermittelt, welche der DUs **62** benötigt werden, auf der Basis der Konfiguration der Texturadressen. Eine Logikreduzierung erfolgt an den registrierten Ausgängen jeder DU und der Multiplexierung des Texturausgangs von jeder DU. Die Zuordnungsaufhebungseinheit **84** übernimmt jetzt die obere und die untere Decodiereinheit **68**, **70** und gibt dieses Mal die dynamisch zugeordneten Ressourcendaten zu den Datenpfaden zurück, die ursprünglich die Texel angefordert haben. Die zum Filtern der vier Lower-Level-Mip-Map-Texel benötigte Arithmetik wird ebenfalls vereinfacht.

[0069] Gemäß der Darstellung ist die Zuordnung der Ausgänge vom Zuordnungsaufheber **84** aufgerufen, aber dies muss nicht unbedingt von der Art und Weise abhängen, in der der Ausgang des trilinearen Filters **64** gehandhabt wird. Dies ist ein ausführungstechnisches Detail, das für die Fachperson klar sein wird.

[0070] Ein weiterer Bereich der Logikoptimierung ist in den Filtern **40** (Fig. 5), die die Lower-Level-Mip-Map erzeugen. Wenn das Kompressionsschema von einem Typ ist, der mit einem Index zum Nachschlagen in einem Codebook arbeitet, wie z.B. im VQ oder Palletisation (siehe oben), dann können in diesem Fall die Texturkomponenten auf eine solche Weise angeordnet werden, dass eine Lower-Mip-Map-Erzeugung erleichtert wird. Das heißt, wenn die Codebook-Einträge durch gleichmäßiges Abtasten der unkomprimierten Textur über ihren Farbbereich erzeugt werden, dann können die Indexe selbst als Eingang in die Mip-Map-Filter benutzt werden, um die neue Lower-Level-Mip-Map zu erzeugen. Der resultierende Index zeigt dann auf das nächste Codebook-Äquivalent der gefilterten Textur. Die Bitbreite der Filterarithmetik an den Indexen (komprimierte Textur) ist weitaus geringer als sie es wäre, wenn der Filter auf die unkomprimierte Textur angewendet worden wäre.

[0071] Eine solche Anordnung ist in Fig. 10 der Zeichnungen illustriert. Sie basiert auf Fig. 7, könnte aber natürlich auch die Modifikation von Fig. 9 beinhalten.

[0072] Fig. 10 zeigt, wie der Filter **66** jetzt in der Dekompressionseinheit **62** aussieht. Jede Dekompressionseinheit **62** hat einen Index-Lookup-Abschnitt **92**, der mit dem Ausgang des Zuordners **82** verbunden ist, um Indexe bereitzustellen, die von dem Filter **66** gefiltert werden, bevor sie an einen Codebook-Lookup-Abschnitt **94** angelegt werden, wo die gefilterten Indexe zum Finden des entsprechenden Codebook-Teils benutzt werden.

[0073] Fig. 11 zeigt schematisch die Beziehung zwischen den komprimierten Code-Werten und den Farben, die sie repräsentieren. Das Diagramm wird als ein Teil des Farbdreiecks angenommen, das alle möglichen Farben auf einer zweidimensionalen Ebene repräsentiert. Zwei Farben C1 und C2 sind als Hauptfarben bezeichnet. Sie definieren die Enden einer Linie, auf der andere Zwischenfarben C11 bis C1i liegen. Diese anderen Farben können daher dadurch gebildet werden, dass ein entsprechender gewichteter Mittelwert der Farben C1 und C2 genommen wird. Die Code-Werte für die Farben C11 bis C1i werden ebenfalls durch Mittelwertbildung der Code-Werte der Farben C1 und C2 definiert.

[0074] Die normale Art und Weise, die Zwischenfarben zu produzieren, besteht darin, mit den komprimierten Code-Werten der Farben C1 und C2 zu beginnen und die tatsächlichen Farbwerte der Farben C1 und C2 mit der Genauigkeit des Systems zu ermitteln, die 16 oder 24 Bit pro Farbe erfordern. Dann wird an den so erhaltenen 16 oder 24 Farbwerten ein gewichteter Mittelwert gebildet. Wir haben jedoch verstanden, dass es, wenn die Beziehung zwischen den Farben und ihren Code-Werten in der soeben beschriebenen Weise definiert wird, möglich ist, die Code-Werte zuerst zu mitteln und dann die so erhaltenen Bemittelten Code-Werte in die nächstliegende verfügbare Farbe umzuwandeln. Dies bedeutet, dass der Mittelwertbildungsvorgang nicht mehr mit 16 oder mehr Datenbits zu erfolgen braucht, sondern jetzt mit den etwa zwei Bits geschehen kann, die zum Definieren der verschiedenen komprimierten Code-Werte beim Gebrauch benötigt werden.

[0075] Diese Lower-Mip-Map-Optimierungstechnik kann auch auf Direct X komprimierte Texturen angewendet werden [Dok. 14]. Hier sind die beiden Grundfarben und die interpolierten Zwischenwerte, auf die die Indexe zeigen, gleichmäßig voneinander beabstandet. Fig. 12 zeigt, wie 2-Bit-Indexe auf 16-Bit-Texturfarben gemappt werden. Es ist ersichtlich, dass die 2-Bit-Indexe selbst anstatt des vollen 16-Bit-Wertes gefiltert werden können. Dies repräsentiert weitaus weniger Logik und somit geringere Gesamtkosten.

[0076] Zusammenfassend ist zu sehen, dass die illustrierte Methode des Erzeugens von Texturdaten für die Verwendung beim Texturieren eines Bildes zunächst das Darstellen von Texturdaten durch arbiträre komprimierte Codes umfasst, in denen gewählte komprimierte Code-Werte Hauptfarben definieren und andere komprimierte Code-Werte Farben definieren, die von gewählten gewichteten Mittelwerten von Hauptfarben gebildet werden können, wobei die entsprechenden Code-Werte auch gewichtete Mittelwerte der Code-Werte der gewählten Hauptfarben sind. Dann wird ein Ausgangstexel von einer Mehrzahl von Eingangstexeln interpoliert, wobei der Interpolationsschritt mit komprimierten Code-Werten erfolgt. Die Code-Werte werden nachfolgend dekomprimiert, so dass sich die eigentlichen Farbwerte ergeben.

[0077] Dieses Verfahren findet auch für andere Zwecke als nur in dem illustrierten Lower-Level-Mip-Map-Generator Anwendung. Es kann überall dort zum Einsatz kommen, wo Zwischenfarben generiert werden sollen, die in komprimierter Form vorliegen, unter der Voraussetzung, dass die komprimierten Codes eine lineare Beziehung haben, die der der tatsächlichen Farben gleich kommt. Insbesondere kann es bei der ursprünglichen Generation der im Speicher **22** befindlichen Mip-Maps angewendet werden.

[0078] Die beschriebenen und illustrierten Ausgestaltungen der Erfindung ergeben verschiedene Verbesserungen gegenüber den bekannten Systemen. Insbesondere wird die Speicherbandbreite für trilineare Filterung effizient genutzt, besonders dann, wenn Texturkompressionscaches verwendet werden. Durch Erzeugen der Lower-Level-Mip-Map-Daten von komprimierten Upper-Level-Mip-Map-Daten wie verlangt, werden Zugriffe über Seitenumbrüche reduziert. Es kann eine Quad-Cache-Anordnung verwendet werden, um ein trilineares gefiltertes Pixel pro Taktperiode zu garantieren. Der Betrieb der Dekompressionslogik wird durch Verwenden von dynamischer Dekompressionsressourcenzuordnung verbessert. Schließlich ergibt die Vereinheitlichung der Lower-Level-Mip-Map-Generation mit der Dekompressionslogik ein System mit relativ geringem Overhead.

[0079] Es ist klar, dass zahlreiche Modifikationen an den beschriebenen und illustrierten Systemen vorgenommen werden können, die lediglich gewählte und derzeit bevorzugte Ausgestaltungen der in den Ansprüchen definierten Erfindung repräsentieren.

Patentansprüche

1. Texturiersystem zur Verwendung in einem dreidimensionalen Abbildungssystem, das Folgendes umfasst:
Speichermittel (**22**) zum Speichern von Mip-Map-Daten in komprimierter Form für die Verwendung beim Texturieren eines Bildes, wobei die Mip-Map-Daten eine hierarchische Serie von Mip-Maps von verschiedenen Leveln von abnehmender Auflösung umfassen;
Eingabemittel (**26**) zum Empfangen von Eingabedaten, die den Typ der benötigten Mip-Map-Daten und den Level des/der Mip-Maps anzeigen, von dem/denen die Daten genommen werden sollen;
Steuermittel (**24**), die mit dem Eingabemittel und dem Speichermittel gekoppelt sind, um die Mip-Map-Daten aus dem Speicher abzurufen, die gemäß den Eingabedaten benötigt werden; und
Cache-Speichermittel (**30**; **60**), die mit dem Steuermittel gekoppelt sind, um Teile von Mip-Map-Daten zu speichern, die aus dem Speicher abgerufen wurden und sich auf einen gewählten Mip-Map-Level beziehen;
trilineares Interpolationsmittel (**34**; **64**), das mit dem Cache-Speichermittel gekoppelt ist, um Mip-Map-Daten von einem Mip-Map-Level zu empfangen und ein Ausgabetexel von Eingabetexeln von zwei Mip-Map-Leveln

zu interpolieren;
Mittel (**36**, **66**) zum Erzeugen von Mip-Maps auf einem tieferen Level, die zwischen dem Cache-Speichermittel und dem trilinearen Interpolationsmittel geschaltet sind, um in der hierarchischen Serie des Mip-Map nächsttiefere Teile des Mip-Map zu erzeugen, deren Mip-Maps im Cache-Speichermittel gespeichert sind; und
Dekompressionsmittel (**32**, **62**) zum Dekomprimieren von Mip-Map-Daten.

2. Texturiersystem nach Anspruch 1, wobei der Eingang des Dekompressionsmittels (**32**; **62**) mit dem Cache-Speichermittel und sein Ausgang mit dem trilinearen Interpolationsmittel und dem Mittel zum Erzeugen von Mip-Maps auf einem tieferen Level verbunden sind.

3. Texturiersystem nach Anspruch 1 oder 2, bei dem vier Cache-Speichermittel (**60**) und vier Dekompressionsmittel (**62**) parallel angeordnet sind.

4. Texturiersystem nach Anspruch 1, 2 oder 3, ferner umfassend Zuweisungsmittel (**82**) zwischen dem Cache-Speichermittel (**60**) und dem Dekompressionsmittel (**62**), um die Ausgänge verschiedener Cache-Speicher ausgewählten Dekompressionsmitteln zuzuordnen.

5. Texturiersystem nach einem der vorherigen Ansprüche, bei dem das Mittel zum Erzeugen von Mip-Maps auf einem tieferen Level vier Interpolatoren umfasst, die auf 16 Texel aus dem im Cache-Speichermittel gespeicherten Mip-Map wirken, so dass vier Texel ab dem nächsttieferen Mip-Map entstehen.

6. Texturiersystem nach Anspruch 1, bei dem die komprimierten Mip-Map-Daten arbiträre komprimierte Codes umfassen, wobei gewählte komprimierte Code-Werte Hauptfarben definieren und andere komprimierte Code-Werte Farben definieren, die durch gewählte gewichtete Mittelwerte von Hauptfarben gebildet werden, wobei die entsprechenden Code-Werte ebenso gewichtete Mittelwerte der Code-Werte der gewählten Hauptfarben sind und das Mittel zum Erzeugen von Mip-Maps auf einem tieferen Level ein Ausgabetexel aus einer Mehrzahl von Eingabetexeln mit den komprimierten Code-Werten interpoliert.

7. Verfahren zum Texturieren für die Anwendung in der dreidimensionalen Abbildung, umfassend die folgenden Schritte:
Speichern von Mip-Map-Daten in komprimierter Form für die Verwendung bei der Texturierung eines Bildes in einem Speicher, wobei die Mip-Map-Daten eine hierarchische Serie von Mip-Maps von verschiedenen Levels von abnehmender Auflösung umfassen;
Empfangen von Eingabedaten, die den Typ von benötigten Mip-Map-Daten und den Level des/der Mip-Map(s) anzeigen, von dem/denen die Daten genommen werden sollen;
Abrufen der benötigten Mip-Map-Daten aus dem Speicher, die gemäß den Eingabedaten benötigt werden; und
Speichern von Teilen von Mip-Map-Daten in einem Speicher, die aus dem Speicher abgerufen wurden und sich auf einen gewählten Mip-Map-Level beziehen; und
Empfangen von Mip-Map-Daten von einem Mip-Map-Level und Interpolieren eines Ausgabetexels von Eingabetexels von zwei Mip-Map-Leveln;
Erzeugen von in der hierarchischen Serie des Mip-Map nächsttieferen Teilen des Mip-Map, deren Mip-Maps im Cache-Speichermittel gespeichert werden; und
Dekomprimieren der Mip-Map-Daten von der Interpolation eines Ausgabetexels.

8. Verfahren zum Texturieren nach Anspruch 7, umfassend die folgenden Schritte:
Darstellen der Mip-Map-Texturdaten mit arbiträren komprimierten Codes, wobei gewählte komprimierte Code-Werte Hauptfarben definieren und andere komprimierte Code-Werte Farben definieren, die anhand von gewählten gewichteten Mittelwerten von Hauptfarben gebildet werden können, wobei die entsprechenden Code-Werte ebenso gewichtete Mittelwerte der Code-Werte der gewählten Hauptfarben sind; und
Verwenden der komprimierten Code-Werte in dem Schritt des Erzeugens von Teilen des in der hierarchischen Serie nächsttieferen Mip-Maps.

Es folgen 9 Blatt Zeichnungen

Anhängende Zeichnungen

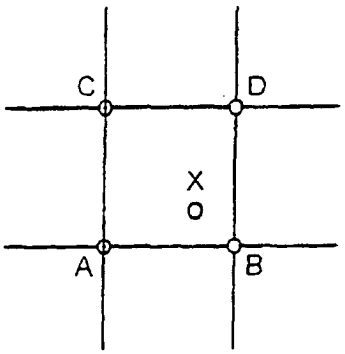


FIG. 1

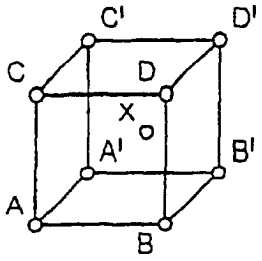


FIG. 2

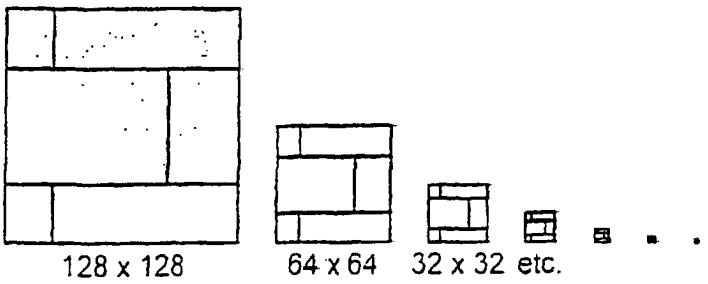


FIG. 3

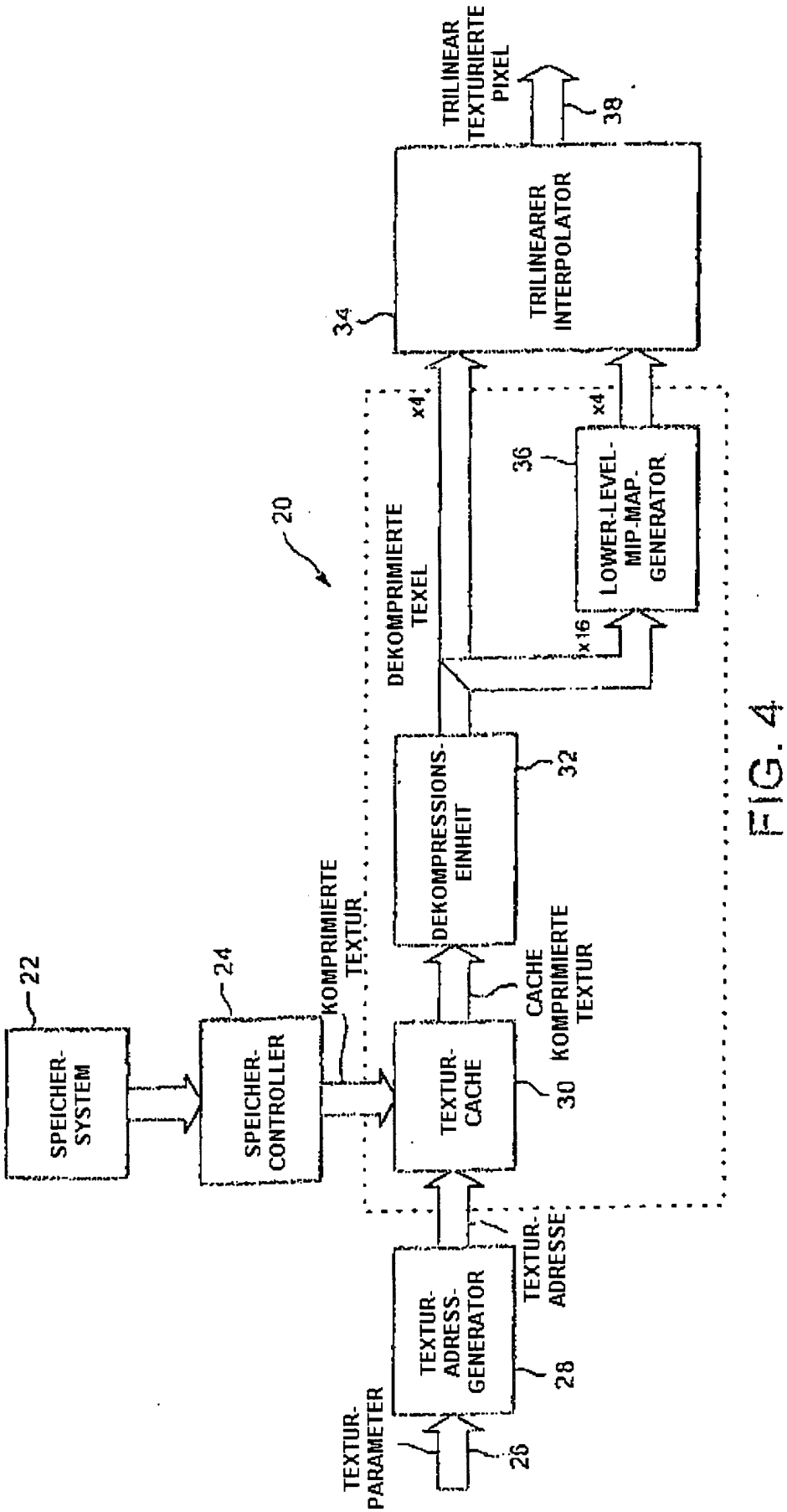

SPEICHER-SYSTEM
22
SPEICHER-CONTROLLER
24
KOMPRIMIERTE TEXTUR
TEXTUR-CACHE
30
CACHE KOMPRIMIERTE TEXTUR
DEKOMPRESSIONS-EINHEIT
32
DEKOMPRIMIERTE TEXEL
x4
x16
LOWER-LEVEL-MIP-MAP-GENERATOR
36
x4
TRILINEARER INTERPOLATOR
34
TRILINEAR TEXTURIERTE PIXEL
38
20
TEXTUR-ADRESS-GENERATOR
28
TEXTUR-ADRESSE
TEXTUR-PARAMETER
26

FIG. 4

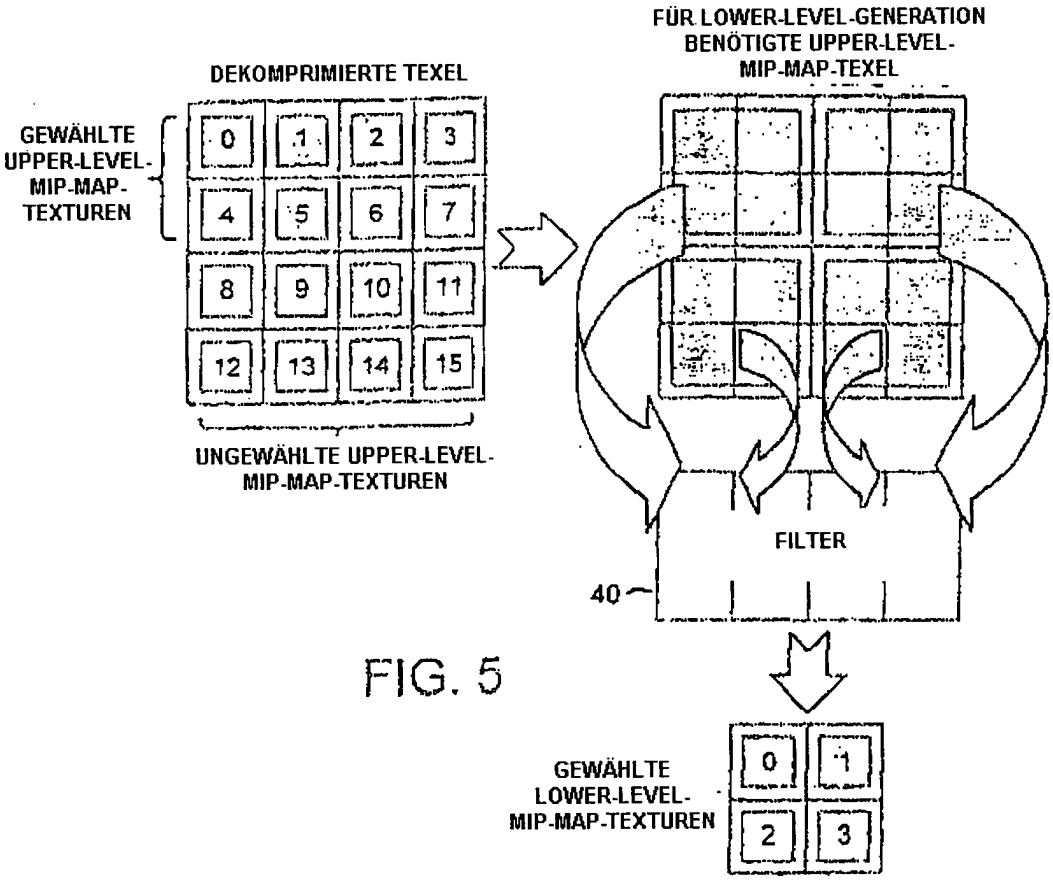

DEKOMPRIMIERTE TEXEL
FÜR LOWER-LEVEL-GENERATION BENÖTIGTE UPPER-LEVEL-MIP-MAP-TEXEL
GEWÄHLTE UPPER-LEVEL-MIP-MAP-TEXTUREN
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
UNGEWÄHLTE UPPER-LEVEL-MIP-MAP-TEXTUREN
FILTER
40
GEWÄHLTE LOWER-LEVEL-MIP-MAP-TEXTUREN
0
1
2
3

FIG. 5

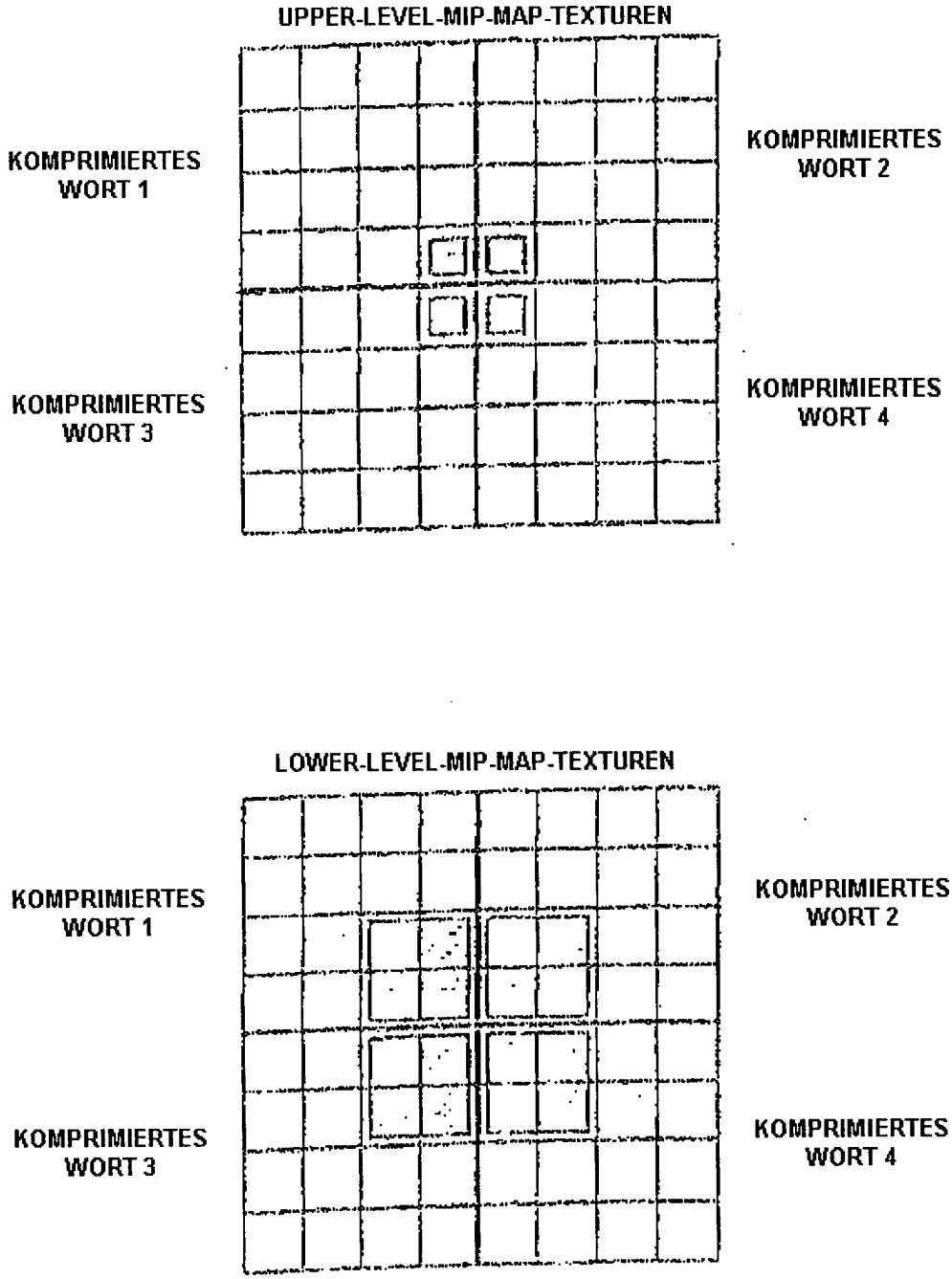
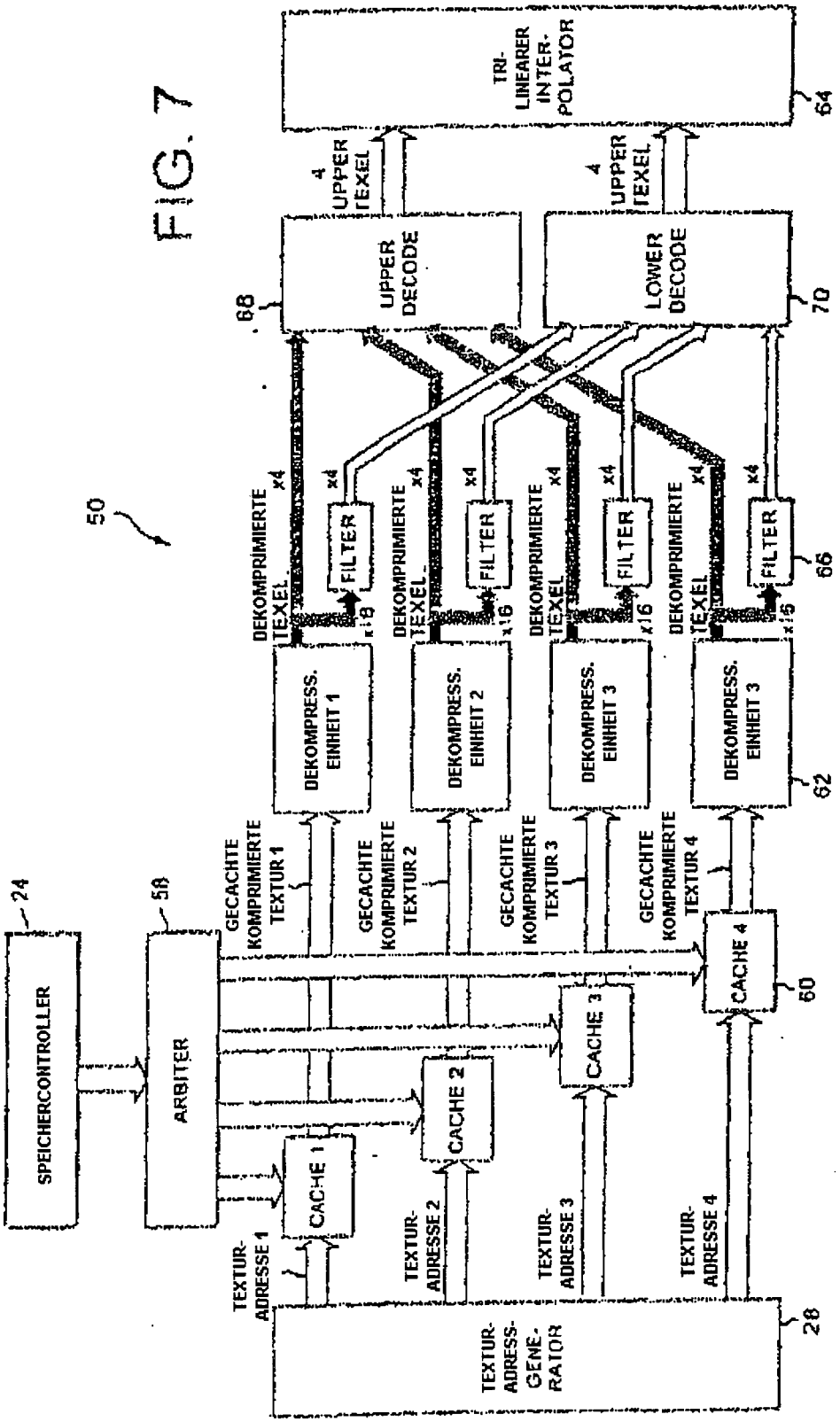
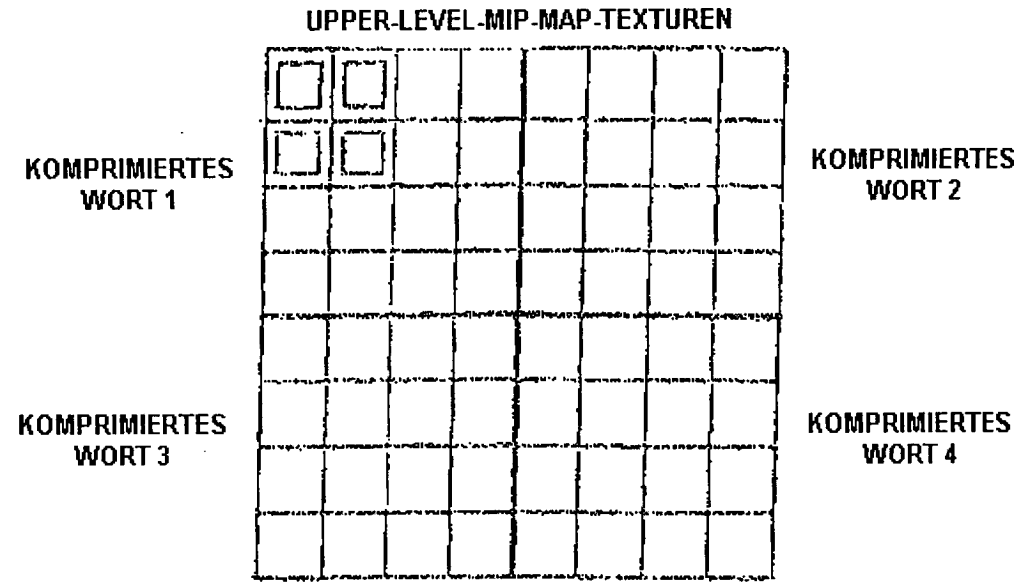

UPPER-LEVEL-MIP-MAP-TEXTUREN
KOMPRIMIERTES WORT 1
KOMPRIMIERTES WORT 2
KOMPRIMIERTES WORT 3
KOMPRIMIERTES WORT 4
LOWER-LEVEL-MIP-MAP-TEXTUREN
KOMPRIMIERTES WORT 1
KOMPRIMIERTES WORT 2
KOMPRIMIERTES WORT 3
KOMPRIMIERTES WORT 4

FIG. 6

FIG. 7


50
24
SPEICHERCONTROLLER
58
ARBITER
GECACHTE KOMPRIMIERTE TEXTUR 1
GECACHTE KOMPRIMIERTE TEXTUR 2
GECACHTE KOMPRIMIERTE TEXTUR 3
GECACHTE KOMPRIMIERTE TEXTUR 4
CACHE 1
CACHE 2
CACHE 3
CACHE 4
60
TEXTUR-ADRESSE 1
TEXTUR-ADRESSE 2
TEXTUR-ADRESSE 3
TEXTUR-ADRESSE 4
TEXTUR-ADRESS-GENE-RATOR
28
DEKOMPRESS. EINHEIT 1
DEKOMPRESS. EINHEIT 2
DEKOMPRESS. EINHEIT 3
DEKOMPRESS. EINHEIT 3
62
DEKOMPRIMIERTE TEXEL x16
FILTER
66
68
UPPER DECODE
LOWER DECODE
70
4 UPPER TEXEL
TRI-LINEARER INTER-POLATOR
64


UPPER-LEVEL-MIP-MAP-TEXTUREN
KOMPRIMIERTES WORT 1
KOMPRIMIERTES WORT 2
KOMPRIMIERTES WORT 3
KOMPRIMIERTES WORT 4

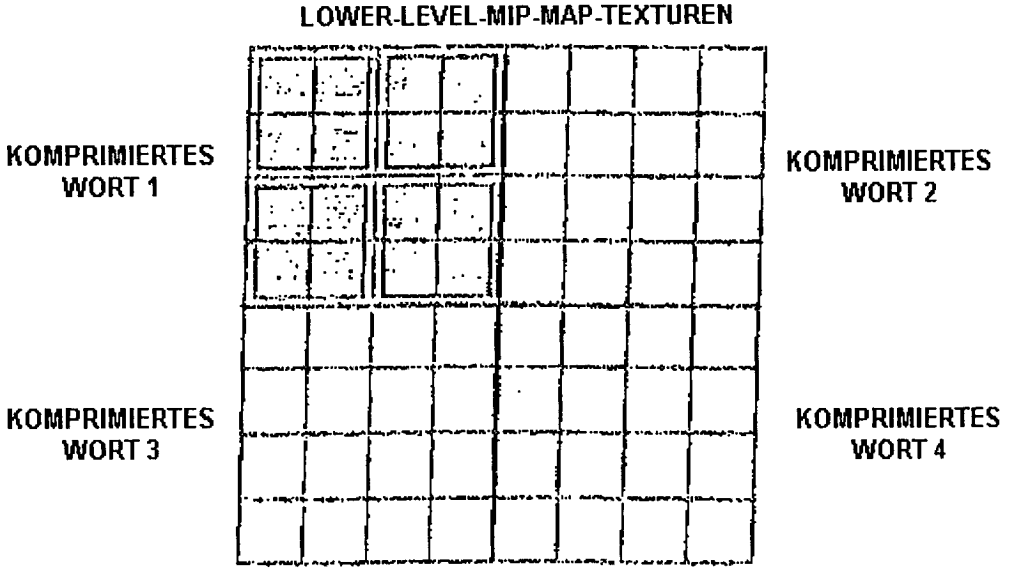

LOWER-LEVEL-MIP-MAP-TEXTUREN
KOMPRIMIERTES WORT 1
KOMPRIMIERTES WORT 2
KOMPRIMIERTES WORT 3
KOMPRIMIERTES WORT 4

FIG. 8

FIG. 9

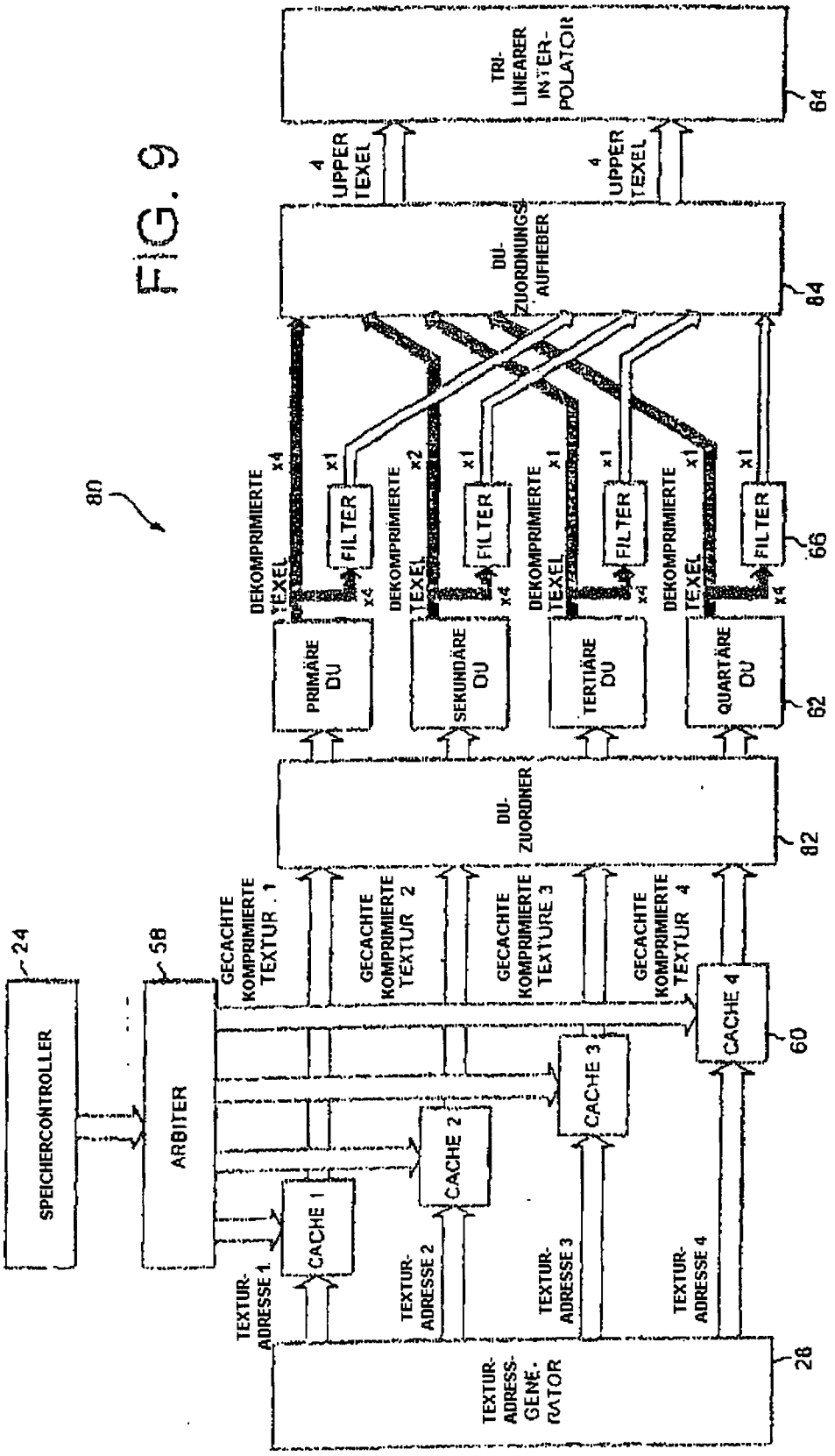
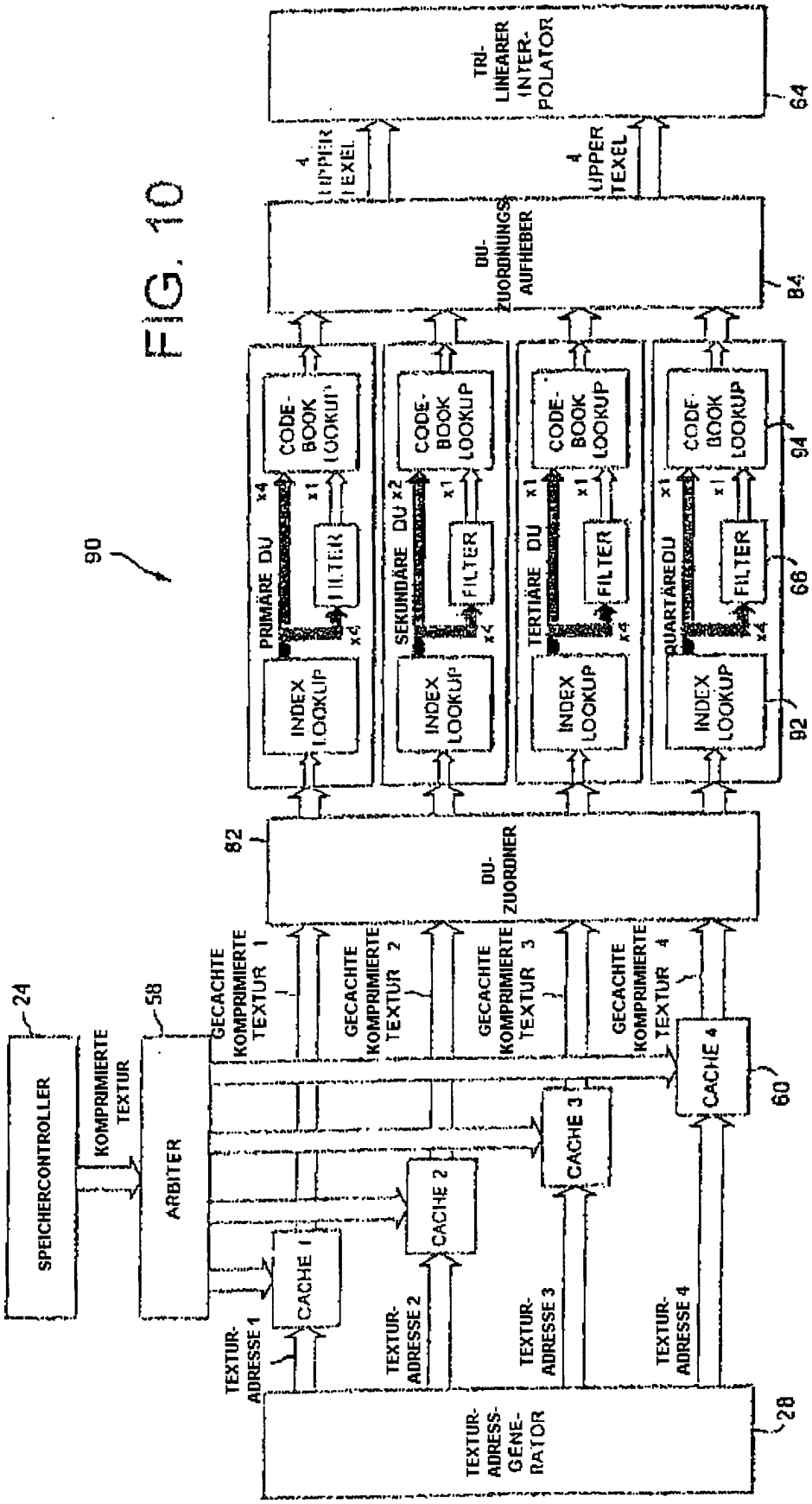

80
24
SPEICHERCONTROLLER
58
ARBITER
GECACHTE KOMPRIMIERTE TEXTUR 1
GECACHTE KOMPRIMIERTE TEXTUR 2
GECACHTE KOMPRIMIERTE TEXTUR 3
GECACHTE KOMPRIMIERTE TEXTUR 4
DU-ZUORDNER
82
PRIMÄRE DU
SEKUNDÄRE DU
TERTIÄRE DU
QUARTÄRE DU
62
DEKOMPRIMIERTE TEXEL x4
FILTER
66
DU-ZUORDNUNGS AUFHEBER
84
4 UPPER TEXEL
TRI-LINEARER INTER-POLATOR
64
CACHE 1
CACHE 2
CACHE 3
CACHE 4
60
TEXTUR-ADRESSE 1
TEXTUR-ADRESSE 2
TEXTUR-ADRESSE 3
TEXTUR-ADRESSE 4
TEXTUR-ADRESS-GENE-RATOR
28

FIG. 10


90
SPEICHERCONTROLLER
24
KOMPRIMIERTE TEXTUR
ARBITER
58
GECACHTE KOMPRIMIERTE TEXTUR 1
GECACHTE KOMPRIMIERTE TEXTUR 2
GECACHTE KOMPRIMIERTE TEXTUR 3
GECACHTE KOMPRIMIERTE TEXTUR 4
DU-ZUORDNER
82
PRIMÄRE DU x4
SEKUNDÄRE DU x2
TERTIÄRE DU x1
QUARTÄRE DU x1
INDEX LOOKUP
CODE-BOOK LOOKUP
FILTER
x1
x4
92
94
66
DU-ZUORDNUNGS AUFHEBER
84
4 UPPER TEXEL
4 UPPER TEXEL
TRI-LINEARER INTER-POLATOR
64
CACHE 1
CACHE 2
CACHE 3
CACHE 4
60
TEXTUR-ADRESSE 1
TEXTUR-ADRESSE 2
TEXTUR-ADRESSE 3
TEXTUR-ADRESSE 4
TEXTUR-ADRESS-GENE-RATOR
28

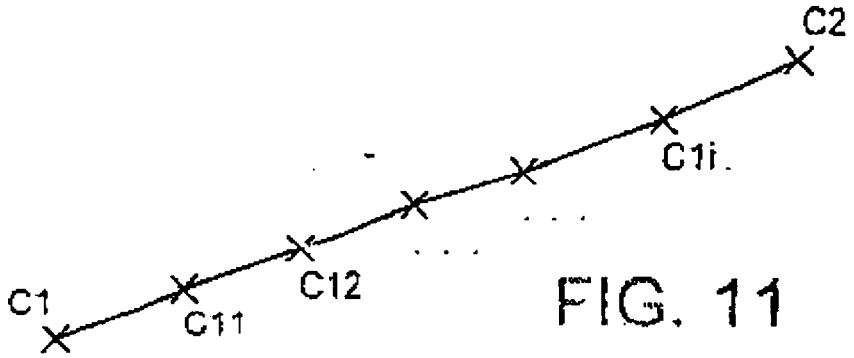

C2
C1i
C12
C11
C1

FIG. 11

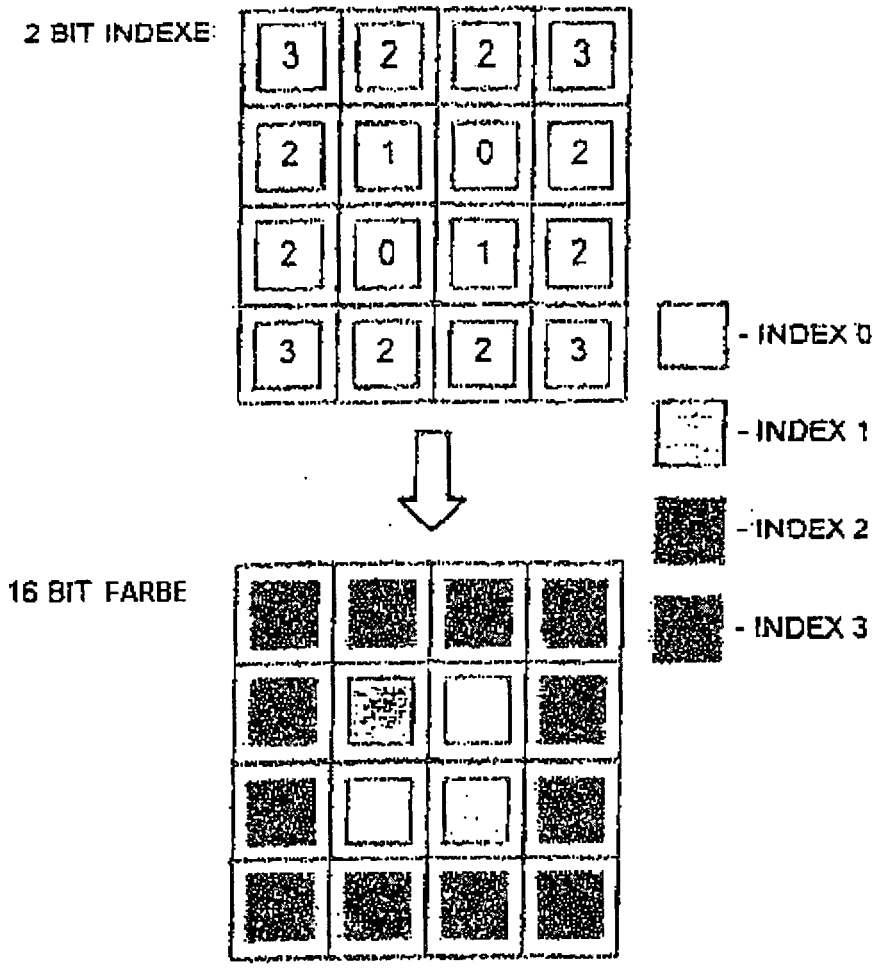

2 BIT INDEXE
3 2 2 3
2 1 0 2
2 0 1 2
3 2 2 3
- INDEX 0
- INDEX 1
- INDEX 2
- INDEX 3
16 BIT FARBE

FIG. 12