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

(10) **DE 696 35 403 T2** 2006.07.27

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 783 154 B1**
(21) Deutsches Aktenzeichen: **696 35 403.9**
(96) Europäisches Aktenzeichen: **96 119 990.8**
(96) Europäischer Anmeldetag: **12.12.1996**
(97) Erstveröffentlichung durch das EPA: **09.07.1997**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **09.11.2005**
(47) Veröffentlichungstag im Patentblatt: **27.07.2006**

(51) Int Cl.[8]: ***G06F 13/10*** (2006.01)
G06F 12/08 (2006.01)
G06F 11/00 (2006.01)
G06T 15/00 (2006.01)

(30) Unionspriorität:
576872 **21.12.1995** **US**

(73) Patentinhaber:
Trepton Research Group, Inc., Santa Clara, Calif., US

(74) Vertreter:
Schwan Schwan Schorer, 80796 München

(84) Benannte Vertragsstaaten:
BE, DE, ES, FR, GB, IE, IT, NL, PT

(72) Erfinder:
Devic, Goran, Austin, Texas 78753, US

(54) Bezeichnung: **Grafikbibliothek auf geteilten Ebenen**

Beschreibung

Gebiet der Erfindung

[0001] Die vorliegende Erfindung bezieht sich auf das Gebiet computergesteuerter grafischer Anzeigesysteme. Im Einzelnen bezieht sich die vorliegende Erfindung auf Softwareprogramme, die eine Schnittstelle zu Hardware-Grafikbeschleunigern bilden.

Hintergrund der Erfindung

[0002] Computergesteuerte Hochleistungs-Grafikdarstellungssysteme hoher Qualität beruhen in großem Umfang auf spezialisierten elektronischen Leiterplatten für die Verarbeitung von grafischen Informationen mit hohen Geschwindigkeiten. Diese spezialisierten elektronischen Leiterplatten werden auch als "Grafikbeschleuniger" oder "Grafikhardwareeinheiten" bezeichnet. Grafikbeschleuniger sind wie beim Stand der Technik bekannt speziell dazu entworfen, Grafikdaten zu verarbeiten, die mit Darstellungsgrundelementen (z.B. Linien, Polygonen, Dreiecken, schattierten oder Alpha-Blending-Dreiecken usw.) verbunden sind, um auf einer Computeranzeigeeinheit ein Bild darzustellen. Die den Grafikbeschleunigern zugeführten Grafikdaten werden in einem hardwareabhängigen Format ("hardwareabhängige Grafikdaten") bereitgestellt, das von dem Grafikbeschleuniger erkannt wird. Diese hardwareabhängigen Grafikdaten werden typischerweise in der Form einer Anzeigeliste im Computerspeicher generiert.

[0003] In einem in mehreren Ebenen arbeitenden Grafiksystem, arbeitet der Grafikbeschleuniger auf der Ebene des untersten Levels, um Pixel zu manipulieren, damit auf der Computeranzeige ein Bild dargestellt wird. Der Grafikbeschleuniger führt Low-Level-Grafikoperationen aus, die Mikrobefehle innerhalb einer Anzeigeliste in das Bild auf einer Computeranzeige umsetzen. In Ebenen höherer Level generieren innerhalb des Grafiksystems ausgeführte Softwareprogramme ("Anwendungen") Anfragen für die Darstellung bestimmter Bilder. Diese Anfragen beinhalten typischerweise eine Liste hardwareunabhängiger Darstellungsgrundelemente (z.B. solche, die das Bild darstellen), die zu Zwischen-High-Level-Grafikbibliotheken und anschließend zu dem Grafikbeschleuniger geführt werden, um dargestellt zu werden. Die High-Level-Grafikbibliotheken unterstützen eine große Anzahl an Grafikbefehlen und Merkmalen und beinhalten typischerweise Prozeduren zur direkten Transformierung der Anfragen der Anwendung zu hardwareabhängigen Anzeigelisten.

[0004] Da die Grafikbeschleuniger Hardwarevorrichtungen sind, sind sie sehr entwurfs- und herstellerspezifisch. Mit anderen Worten arbeiten unterschiedliche Grafikbeschleuniger unter Verwendung unterschiedlicher Grafikdatenformate (z.B. unterschiedliche Datenstrukturen, verschiedene Grafikdarstellungscodes, unterschiedliche Speicherpartitionen und -stellen usw.) und unter Verwendung unterschiedlicher Cachezuweisungen.

[0005] Fig. 1A stellt ein beim Stand der Technik bestehendes grafisches Anzeigesystem **5** gemäß der obigen Ausführung dar. Das System **5** beinhaltet ein High-Level-Anwendungsprogramm **10**, das in einem digitalen Computersystem ausführbar ist. Das Anwendungsprogramm **10** ist mit High-Level-Grafikbibliotheken **12** und **14** verbunden, um Grafikanzeigeanfragen von der Anwendung **10** in hardwareabhängige Anzeigelisten zu übersetzen. Zwei wohlbekannte Grafikbibliotheken sind die Bibliothek **12** der dreidimensionalen "Dynamic Device Driver"-Schnittstelle ("3D DDI") und die "Offene Grafikbibliothek" **14** ("Open Graphics Library bzw. "Open GL"). Diese Bibliotheken **12** und **14** enthalten eine Anzahl an hardwarespezifischen Darstellungsroutinen ("Prozeduren"), die über eine Schnittstelle **16** direkt mit der Grafikhardwareeinheit **18** (z.B. einer Beschleunigerplatine) verbunden sind. Die Schnittstelle **16** lässt die Hardwareeinheit **18** nicht transparent werden, sondern dient lediglich dazu, das Kommunikationsprotokoll zwischen den Bibliotheken **12** und **14** und der Hardwareeinheit **18** zu vereinfachen. Die Prozeduren der Bibliotheken **12** und **14** stellen Grafikdarstellungsanfragen von dem Anwendungsprogramm **10** als Eingang bereit. Die Grafikhardwareeinheit **18** ist mit einem Computerbildschirm **20** verkoppelt, um auf ihm Bilder darzustellen. Obgleich als während der Kompilierung und des Verlinkens abgetrennt dargestellt, sind die erforderlichen Prozeduren der Bibliotheken **12** und **14** aus Fig. 1A typischerweise mit eingeschlossen, um die Befehle der High-Level-Anwendung **10** auszubilden. Ebenfalls kann die Anwendung **10** Befehle beinhalten, die von einer Anzahl an anderen Bibliotheken neben der Grafikbibliothek **12** und der Grafikbibliothek **14** resultieren.

[0006] Die Prozeduren der Grafikbibliotheken **12** und **14** ermöglichen eine Kommunikation der High-Level-Anwendung **10** (unter Verwendung von hardwareunabhängigen Datenstrukturen) mit der Grafikhardwareeinheit **18**, indem sehr strukturierte und hardwareabhängige Kommunikationsprotokolle und Datenstrukturen generiert werden. Die Grafikbibliotheken **12** und **14** beinhalten Sätze von hardwareabhängigen und auf sehr

hohen Levels arbeitenden Softwareroutinen, die zwar hardwareabhängig sind, aber für die Anwendung **10** eine hardwaretransparente Schnittstelle bereitstellen. Da die Bibliotheken **12** und **14** High-Level-Bibliotheken sind, unterstützen sie eine große Vielzahl von komplexen Grafikmerkmalen und Anzeigeoptionen. Um sowohl die Bibliothek **12** wie die Bibliothek **14** so zu implementieren, dass sie irgendeine bestimmte Hardwareeinheit **18** unterstützen, müssen sämtliche Bibliothekfunktionen und -merkmale in dem jeweiligen Format der gewählten Hardwareeinheit **18** implementiert werden. Daher müssen diese High-Level-Grafikbibliotheken **12** und **14** umfangreich umgestaltet und für jede unterschiedliche Hardwareeinheit **18**, die sie unterstützen, umgeschrieben werden. Für den Gestalter von Grafiksystemen ist dies unerwünscht, da ein großes Ausmaß an Softwareumgestaltung und Umschreiben notwendig ist, damit jede Bibliothek **12** und **14** unterschiedliche Hardwareeinheiten **18** unterstützt. Die Bereitstellung eines Grafiksystems wäre erwünscht, das einfacher an Veränderungen und Variationen in der Grafikhardwareeinheit **18** angepasst werden kann. Im Einzelnen wäre die Bereitstellung eines Softwaresystems erwünscht, das keine Umgestaltung und kein Umschreiben der High-Level-Grafikbibliotheken **12** und **14** bei einer Anwendung von unterschiedlichen Grafikhardwareeinheiten **18** benötigt.

[0007] Auf Fig. 1A und Fig. 1B Bezug nehmend ist in Fig. 1B ein Ablaufdiagramm eines computerimplementierten Verfahrens **30** vom Stand der Technik für die Verarbeitung von Darstellungsgrundelementen illustriert. Dieses Verfahren **30** ist innerhalb eines beim Stand der Technik bestehenden computergesteuerten grafischen Anzeigesystems **5** (Fig. 1A) implementiert. Das Verfahren **30** beginnt mit einem Block **32**, an dem das High-Level-Anwendungsprogramm **10** die Darstellung einer Datenstruktur abfragt, die ein individuelles Darstellungsgrundelement (z.B. Linie, Polygon, Dreieck usw.) an der Anzeige **20** repräsentiert. Bei einem Block **34** übermittelt die Anwendung **10** die das Darstellungsgrundelement repräsentierende hardwareunabhängige Datenstruktur zu einer Prozedur der Grafikbibliothek **12** oder **14**. An einem Block **36** setzt die Grafikbibliothek **12** oder **14** die Datenstruktur des Darstellungsgrundelements in einen Satz von Low-Level-Mikrobefehlen um, die für die Hardwareeinheit **18** spezifisch sind. Diese Mikrobefehle sind häufig Teil einer "Anzeigeliste", die von der Hardwareeinheit **18** ausgelesen werden kann. Bei einem Block **38** greift die Hardwareeinheit **18** auf die Anzeigeliste zu, um das Darstellungsgrundelement an dem Bildschirm **20** darzustellen. Anschließend werden nachfolgende von dem Anwendungsprogramm **10** stammende Grundelemente auf eine ähnliche Weise verarbeitet, da für die Darstellung eines vollständigen Bilds typischerweise eine Mehrzahl von Darstellungsgrundelementen erforderlich ist.

[0008] Das Verfahren **30** von Fig. 1B verwendet die Speicherressourcen innerhalb des Grafiksystems **5** nicht auf effiziente Weise, da jedes Grundelement von dem Block **32** zwischen der High-Level-Anwendung **10** und der Hardwareeinheit **18** (z.B. zwischen den Blöcken **32** und **38**) seriell verarbeitet wird. Im Einzelnen werden bei dem Darstellungsverfahren eines einzelnen Darstellungsgrundelements zuerst Prozeduren innerhalb der Anwendung **10** und anschließend Prozeduren der Grafikbibliothek (**12** oder **14**) ausgeführt, wobei dieses Verfahren für nachfolgende Grundelemente wiederholt wird. Dieser Umstand bewirkt das Auftreten einer Disjunktion in den Code- und Datencaches, da unterschiedliche Informationen und Befehle durch diese Cacheeinheiten geleitet werden, während zugleich die Prozeduren von (1) der High-Level-Anwendung **10** und anschließend (2) Prozeduren der Grafikbibliothek **12** oder **14** ausgeführt werden. Dies führt zu dem Auftreten von vielen Datencachefehlgriffen und Codecachefehlgriffen während der Darstellung eines Satzes von Darstellungsgrundelementen. Somit wäre die Bereitstellung eines Grafikdarstellungsverfahrens vorteilhaft, das bei der Darstellung eines Bildes, das aus einem Satz von aus dem Anwendungsprogramm **10** resultierenden Darstellungsgrundelementen besteht, auf effizientere Weise arbeitet.

[0009] Dementsprechend stellt die vorliegende Erfindung ein Grafiksystem bereit, das auf einfache Weise an unterschiedliche Hardwareeinheiten angepasst werden kann, indem keine Umgestaltung oder kein Umschreiben der High-Level-Grafikbibliothek notwendig ist. Weiterhin stellt die vorliegende Erfindung ein Grafiksystem bereit, das die Speicherressourcen auf effiziente Weise benutzt, um ein Bild aus einer Mehrzahl von Darstellungsgrundelementen darzustellen, die von einem Anwendungsprogramm stammen. Bei der nachfolgenden Erläuterung werden diese und weitere Vorteile der vorliegenden Erfindung deutlich werden.

Zusammenfassung der Erfindung

[0010] Gemäß der vorliegenden Erfindung stellt eine hardwareabhängige Low-Level-Grafikbibliothek eine Schnittstelle zwischen einer High-Level-Grafikbibliothek (die vorzugsweise hardwareunabhängig ist) und einer Grafikbeschleuniger-Hardwareeinheit bereit. Die hardwareunabhängige Low-Level-Bibliothek stellt eine auf relativ niedrigem Level arbeitende Schnittstelle bereit, die direkt mit dem Grafikbeschleuniger kommuniziert und nur eine relativ kleine Menge an Umschreiben benötigt, um sich an unterschiedliche Grafikhardwareeinheiten anzupassen, während eine nur kleine oder gar keine Veränderung der hardwareunabhängigen High-Level-Grafikbibliotheken notwendig ist. Die hardwareabhängigen Low-Level-Bibliothekprozeduren stellen eine

Qualitätszuweisung bereit, die zwischen einer Verarbeitung mit geringer Geschwindigkeit und einer hochqualitativen Bilddarstellung sowie einer mit höherer Geschwindigkeit erfolgender Bilddarstellung in geringer Qualität einstellbar ist. Die hardwareabhängigen Low-Level-Bibliothekprozeduren führen eine Stapelverarbeitung aus, indem ein Feld von Stapelverarbeitungszellen empfangen wird, wobei jede Stapelverarbeitungszelle ein getrenntes Grundelement aufweist. Das Feld kann bei einer Einstellung den hardwareabhängigen Low-Level-Bibliothekprozeduren überreicht und anschließend sequenziell verarbeitet werden. Diese Konfiguration stellt sicher, dass während der Parametrisierung des Feldes (z.B. einer in einen Standardcodecache von z.B. 8 K passenden Parametrisierungsroutine) keine Befehlscache-Fehlgriffe und nur wenige Datencachefehlgriffe auftreten. Weiterhin ermöglichen die hardwareabhängigen Low-Level-Bibliothekprozeduren eine automatische Umsetzung zwischen unterschiedlichen Texture-Mapping-Datenformaten, sodass entweder ein RGB-Alpha-Format oder ein Format benutzt werden kann, das einen Index in die Farbpalette verwendet. Durch die Bereitstellung einer Low-Level-Schnittstelle ermöglicht die vorliegende Erfindung ein System, das an unterschiedliche Hardware-Grafikbeschleuniger einfach angepasst werden kann, ohne dass Modifizierungen der Grafikbibliotheken notwendig werden.

[0011] Im Einzelnen beinhalten die Ausführungsformen der vorliegenden Erfindung ein computergesteuertes grafisches Anzeigesystem mit folgenden Komponenten: einem mit einem Bus gekoppelten Prozessor; einer mit dem Prozessor zusammenarbeitenden Speichereinheit zum Speichern von Informationen; einer Hardware-Grafikeinheit zur Aufnahme von hardwareabhängigen Mikrobefehlen von einer in der Speichereinheit gespeicherten Anzeigeliste zum Erzeugen eines Bilds auf einem Bildschirm; einer High-Level-Grafikbibliothek, die von dem Prozessor ausgeführte hardwareunabhängige Grafikdarstellungsprozeduren aufweist, wobei die hardwareunabhängigen Grafikdarstellungsprozeduren zur Verarbeitung von Grafikdarstellungsanfragen von einer High-Level-Anwendung zur Generierung von hardwareunabhängigen Ausgangsdatenstrukturen Grafikoperanden beinhalten; und einer hardwareabhängigen Low-Level-Grafikbibliothek, die von dem Prozessor zur Verarbeitung der hardwareunabhängigen Ausgangsdatenstrukturen ausgeführt wird, um daraus die Mikrobefehle für die Hardware-Grafikeinheit zu erzeugen, wobei die High-Level-Grafikbibliothek zu einer Vielzahl von unterschiedlichen Hardware-Grafikeinheiten ohne eine Umgestaltung kompatibel ist und wobei die hardwareunabhängigen Ausgangsdatenstrukturen von der High-Level-Grafikbibliothek ein Feld von Stapelverarbeitungszellen aufweisen, wobei jede Stapelverarbeitungszelle eine getrennte auszuführende Grafikoperation darstellt, und wobei das Feld von Stapelverarbeitungszellen zu der hardwareabhängigen Low-Level-Grafikbibliothek geführt wird, um dort zur Generierung der Mikrobefehle nacheinander verarbeitet zu werden.

[0012] Weitere Ausführungsformen beinhalten weiterhin das oben Gesagte, wobei die hardwareabhängige Low-Level-Grafikbibliothek Parametrisierungsprozeduren zur Verarbeitung von Polygon-Grundelementen, Sätze von Grafiklinien und Sätze von Grafikpunkten aufweist und wobei die Parametrisierungsprozeduren weiterhin für eine Verarbeitung von Bitleveltransfers, Füllungen, und Umsetzungen zwischen Texture-Map-Formaten vorgesehen sind.

[0013] Weitere Ausführungsformen beinhalten weiterhin das oben Gesagte, wobei die hardwareabhängige Low-Level-Grafikbibliothek zusätzlich eine von dem Prozessor ausgeführte Leistungs-/Qualitäts-Einstellprozedur aufweist, um die Darstellungsleistungsrate und entsprechend die Darstellungsqualität des auf dem Bildschirm angezeigten Bildes einzustellen. Weitere Ausführungsformen beinhalten zusätzlich ein Verfahren zum Erzeugen einer Anzeigeliste gemäß der obigen Ausführung.

Kurze Beschreibung der Zeichnungen

[0014] Fig. 1A illustriert ein beim Stand der Technik vorliegendes grafisches Anzeigesystem mit hardwareabhängigen Versionen von Grafikbibliotheken (z.B. 3D-DDI und OPEN GL).

[0015] Fig. 1B stellt ein Ablaufdiagramm eines Verfahrens vom Stand der Technik für die Darstellung von Darstellungsgrundelementen dar.

[0016] Fig. 2 ist ein Blockdiagramm eines Computersystems für ein computergesteuertes grafisches Anzeigesystem der vorliegenden Erfindung.

[0017] Fig. 3 illustriert die Ebenen eines computergesteuerten grafischen Anzeigesystems der vorliegenden Erfindung mit hardwareunabhängigen Versionen von High-Level-Grafikbibliotheken (z.B. 3D-DDI und OPEN GL).

[0018] Fig. 4 ist ein Datenablaufdiagramm und illustriert die Komponenten des computergesteuerten grafi-

schen Anzeigesystems der vorliegenden Erfindung einschließlich verschiedener Ebenen und dem Grafikdaten/Informationsfluss zwischen den Ebenen.

[0019] Fig. 5 stellt Komponenten der hardwareabhängigen Low-Level-Grafikbibliothek (HDGL oder "Verbindungs"-Bibliothek) der vorliegenden Erfindung dar.

[0020] Fig. 6 ist eine Logikdarstellung eines Stapelverarbeitungsfeldes gemäß der vorliegenden Erfindung.

[0021] Fig. 7 ist ein Verfahrensablaufdiagramm eines Verfahrens der vorliegenden Erfindung für eine effiziente Darstellung von Darstellungsgrundelementen auf einem Bildschirm unter Verwendung eines Feldes von Stapelverarbeitungszelden.

[0022] Fig. 8A illustriert die Inhalte eines Datencache-Speichers und Inhalte eines Code- oder Befehlscache-Speichers während einer ersten Verarbeitungsphase eines Stapelverarbeitungsfeldes gemäß der vorliegenden Erfindung.

[0023] Fig. 8B ist eine Illustration der Inhalte eines Datencache-Speichers und der Inhalte eines Code- oder Befehlscache-Speichers während einer zweiten Verarbeitungsphase eines Stapelverarbeitungsfeldes gemäß der vorliegenden Erfindung.

[0024] Fig. 8C illustriert die Inhalte eines Datencache-Speichers und Inhalte eines Code- oder Befehlscache-Speichers während einer dritten Verarbeitungsphase eines Stapelverarbeitungsfeldes gemäß der vorliegenden Erfindung.

[0025] Fig. 9A stellt ein Qualitäts-/Leistungs-Steuerfeld gemäß der vorliegenden Erfindung dar.

[0026] Fig. 9B ist ein Ablaufdiagramm eines Verfahrens der vorliegenden Erfindung zum Einstellen der Darstellungsqualität gegenüber der Darstellungsleistung auf der Basis der Einstellungen des Qualitäts-/Leistungs-Steuerfeldes.

[0027] Fig. 10A ist eine grafische Darstellung eines Grafikelements, das unter Verwendung einer linearen Unterteilung unterteilt ist.

[0028] Fig. 10B ist eine grafische Darstellung eines Grafikelements, das unter Verwendung einer perspektivischen Unterteilung unterteilt ist.

[0029] Fig. 10C illustriert einen Überlappungsbereich zwischen zwei dreieckigen Polygonen.

[0030] Fig. 11 ist ein Ablaufdiagramm eines Verfahrens der vorliegenden Erfindung zum Übersetzen zwischen Farbmodi für Texture-Maps.

[0031] Fig. 12 ist ein Ablaufdiagramm eines hardwareabhängigen Verfahrens der vorliegenden Erfindung zum Aufbau einer Anzeigeliste auf der Basis von Grafikdateninformationen, die in einer Stapelverarbeitungszelle eines Stapelverarbeitungsfeldes gespeichert sind.

Beschreibung der bevorzugten Ausführungsformen

[0032] In der folgenden ausführlichen Beschreibung der vorliegenden Erfindung werden zahlreiche spezifische Einzelheiten aufgeführt, um ein besseres Verständnis der vorliegenden Erfindung zu ermöglichen. Allerdings versteht sich für den Fachmann, dass die vorliegende Erfindung auch ohne diese spezifischen Einzelheiten oder unter Verwendung alternativer Elemente oder Verfahren angewendet werden kann. In anderen Fällen sind wohlbekannte Verfahren, Prozeduren, Komponenten und Schaltungen nicht ausführlich beschrieben worden, um die Aspekte der vorliegenden Erfindung nicht unnötig unverständlich ausfallen zu lassen.

Notation und Terminologie

[0033] Gewisse Bereiche der nachfolgenden ausführlichen Beschreibungen erfolgen unter Bezugnahme auf Begriffe wie Prozeduren, Logikblöcke, Verarbeitung und weitere symbolische Repräsentationen von Operationen an Datenbits innerhalb eines Computerspeichers. Diese Beschreibungen und Repräsentationen sind diejenigen Mittel, die für den Fachmann auf dem Gebiet der Datenverarbeitung verwendet werden, um ihr Fach-

gebiet anderen Fachleuten am effektivsten erläutern zu können. Eine Prozedur, ein Logikblock, ein Verfahren usw. wird hier und im allgemeinen als eine in sich widerspruchsfreie Sequenz von Schritten oder Befehlen betrachtet, die zu einem erwünschten Ergebnis führt. Die Schritte sind solche, die physikalische Manipulationen physikalischer Größen erfordern. Üblicherweise, jedoch nicht notwendigerweise nehmen diese physikalischen Manipulationen die Form von elektrischen oder magnetischen Signalen an, die in einem Computersystem gespeichert, übertragen, kombiniert, verglichen und anderweitig manipuliert werden können. Aus Gründen der Einfachheit und mit Bezug auf die allgemeine Sprachverwendung werden diese Signale mit Bezug auf die vorliegende Erfindung als Bits, Werte, Elemente, Symbole, Buchstaben, Begriffe, Zahlen, oder ähnliches bezeichnet.

[0034] Es sollte jedoch berücksichtigt werden, dass alle diese Begriffe als auf physikalische Manipulationen und Größen referierende Begriffe zu interpretieren sind und lediglich den gebräuchlichen Jargon darstellen. Daher sind diese Begriffe angesichts der allgemein beim Stand der Technik verwendeten Terminologie tief gehender zu interpretieren. So lange dies in den folgenden Erörterungen nicht spezifisch anders angegeben ist, versteht sich, dass die in den Erläuterungen der vorliegenden Erfindung verwendeten Begriffe wie z.B. "Verarbeitung", "Rechnung", "Berechnung", "Bestimmung", "Darstellung" oder ähnliches sich auf den Vorgang und die Verarbeitungen eines Computersystems oder einer ähnlichen elektronischen Berechnungsvorrichtung beziehen, das/die Daten manipuliert und überträgt. Die Daten werden als physikalische (elektronische) Größen in den Registern und Speichereinheiten des Computersystems repräsentiert und zu anderen Daten transformiert, die ähnlich dazu als physikalische Größen innerhalb der Speichereinheiten oder Register des Computersystems oder in anderen derartigen Informationsspeicherungs-, Informationsübertragungs- oder -anzeigevorrichtungen repräsentiert sind.

Abschnitt I

Computersystem

[0035] Ein Anwendungsprogramm (**210** von Fig. 3), High-Level-Grafikbibliotheken **220** und **230** und eine hardwareabhängige Low-Level-Grafikbibliothek ("HDGL") **240** der vorliegenden Erfindung bestehen aus ausführbaren Computerbefehlen, die in einem computergesteuerten grafischen Anzeigesystem der vorliegenden Erfindung gespeichert sind. Diese Elemente werden nachstehend beschrieben werden. Fig. 2 illustriert ein exemplarisches Computersystem **112**, das als ein Teil eines computergesteuerten grafischen Anzeigesystems gemäß der vorliegenden Erfindung verwendet wird. Das Computersystem **112** von Fig. 2 versteht sich lediglich als exemplarisch und die vorliegende Erfindung kann in einer Anzahl an unterschiedlichen Computersystemen einschließlich Allzweck-Computersystemen, integrierten Computersystemen und speziell für die Grafikdarstellung ausgelegten Computersystemen angewendet werden, wobei diese Systeme die gleichen Elemente aufweisen können, die wie in Fig. 2 illustriert auf die gleiche Weise untereinander verbunden sind.

[0036] Das Computersystem **112** von Fig. 2 beinhaltet einen Adressen-/Daten-Bus **100** zur Übertragung von Informationen; eine mit dem Bus **100** gekoppelte Zentralprozessoreinheit **101** zur Verarbeitung von Informationen und Befehlen; einen Lese/Schreib-Speicher **102** (z.B. Direktzugriffsspeicher oder ein anderer Lese/Schreib-Speicher wie z.B. FLASH-Speicher usw.), der mit dem Bus **100** zum Speichern von Informationen und Befehlen für den Zentralprozessor **101** gekoppelt ist; sowie einen Nurlese-Speicher **103**, der mit dem Bus **100** zum Speichern von statischen Informationen und Befehlen für den Prozessor **101** gekoppelt ist. Das System **112** beinhaltet eine Datenspeichervorrichtung **104** (z.B. eine magnetische oder optische Platte und Plattenantrieb), die mit dem Bus **100** zum Speichern von Informationen und Befehlen gekoppelt ist. Ebenfalls beinhaltet das System **112** eine Anzeigevorrichtung **105**, die mit dem Bus **100** verkoppelt ist (oder die wahlweise über den Bus **100a** direkt an die Hardwareeinheit **250** gekoppelt sein kann), um einem Computeranwender Informationen (z.B. Darstellungsgrundelemente) darzustellen. Wahlweise kann das System **112** eine alphanumerische Eingabevorrichtung **106** (z.B. eine Vorrichtung einschließlich alphanumerischer und Funktionstasten) aufweisen, die an den Bus **100** gekoppelt ist, um Informationen und Befehlsauswahlen zu dem Zentralprozessor **101** zu übertragen. Wahlweise kann das System **112** eine Cursor-Steuervorrichtung **107** aufweisen, die mit dem Bus **100** verkoppelt ist, um Anwendereingangsinformationen und Befehlsauswahlen zu dem Prozessor **101** zu übertragen. Optional kann das System **112** eine an den Bus **100** gekoppelte Signalerzeugungsvorrichtung **108** für die Übertragung von Befehlsauswahlen zu dem Prozessor **101** beinhalten. Der Prozessor **101** enthält einen Befehls- oder Codecache **102a** und einen Datencache **102b** (z.B. speziell angeordnetes RAM).

[0037] Die in dem Computersystem **112** der vorliegenden Erfindung verwendete Anzeigevorrichtung **105** von Fig. 2 kann eine Flüssigkristall-, eine Kathodenstrahlröhren- oder eine andere Anzeigevorrichtung sein, die ge-

eignet ist, für den Anwender erkennbare Grafikbilder und alphanumerische Zeichen zu generieren.

[0038] Die optionale Cursor-Steuervorrichtung **107** ermöglicht es, dass dem Computeranwender die zweidimensionale Bewegung eines sichtbaren Symbols (Zeiger) auf einem Bildschirm oder einer Anzeigevorrichtung **105** dynamisch signalisiert wird. Beim Stand der Technik sind viele Implementierungen der Cursor-Steuervorrichtung bekannt und schließen einen Trackball, eine Maus, ein Touchpad, Joystick oder spezielle Tasten an der alphanumerischen Eingabevorrichtung **105** ein, die eine Bewegung in einer gegebenen Richtung oder eine Verlagerungsweise signalisieren können. Es versteht sich, dass die Cursor-Anordnung **107** auch über den Eingang der Tastatur unter Verwendung spezieller Tasten- und Tastenfolgenbefehlen geführt und/oder aktiviert werden kann. Alternativ dazu kann der Cursor über den Eingang von einer Anzahl an speziell dazu ausgelegten Cursorlenkvorrichtungen wie oben beschrieben geführt und/oder aktiviert werden. Ebenfalls an den Bus **100** oder wahlweise (über den Bus **100a**) direkt an die Anzeigevorrichtung **105** gekoppelt ist eine Grafikhardware-(z.B. Grafikbeschleuniger)-Einheit **250** für eine Hochgeschwindigkeitsgrafikdarstellung vorgesehen. Die Grafikhardwareeinheit **250** kann ebenfalls Video- und anderen Speicher **102'** aufweisen (z.B. RAM zum Speichern von Anzeigelisten und/oder registrierten Texture-Maps).

[0039] Fig. 2 illustriert, dass die hardwareabhängige Low-Level-Grafikbibliothek (HDGL) **240** der vorliegenden Erfindung in dem RAM **102**, dem ROM **103** und dem Speicher **104** gespeichert werden. Im Betrieb kann ein Teil der HDGL **240** auch in einem Codecache **102a** gespeichert werden.

[0040] Fig. 3 ist eine Logikdarstellung der funktionalen Ebenen eines computergesteuerten grafischen Anzeigesystems **200** gemäß der vorliegenden Erfindung. Abgesehen von der Hardwareeinheit **250** und der Anzeigeeinheit **105** sind die restlichen Elemente von Fig. 3 als ausführbare Befehle innerhalb des Computersystems **112** (Fig. 2) implementiert und können in dem RAM **102**, ROM **103** oder in dem Speicher **104** gespeichert werden und im Betrieb kann ein Teil der HDGL **240** auch in dem Codecache **102a** abgespeichert werden. Die High-Level-Anwendung **210** (z.B. ein Simulator, ein Entwurfswerkzeug, eine Multimedia-Anwendung, eine medizinische Bilddarstellungsanwendung, ein Spiel usw.) beinhaltet Routinen, die eine Erzeugung von Bildern auf dem Bildschirm **105** erfordern. Die Bilder sind aus Darstellungsgrundelementen zusammengesetzt (Punkte, Linien, Polygone, schattierte Polygone, überlagerte Polygone usw.). Die Routinen der Anwendung **210** greifen auf Grafikdarstellungsprozeduren der High-Level-Grafikbibliotheken **220** (3D-DDI) und/oder **230** (OPEN GL) zu, indem angefordert wird, dass bestimmte Darstellungsgrundelemente dargestellt werden, und liefern hardwareunabhängige Grafikstrukturen, die die Darstellungsgrundelemente repräsentieren. Obgleich in der Vergangenheit diese Grafikbibliotheken **220** und **230** hardwareunabhängige Eingänge benutzt haben, waren ihre Ausgänge hardwareabhängig und erforderten spezialisierte Implementierungen für jede unterstützte Hardwareeinheit **250**. Die Benutzung von hardwareabhängigen Prozeduren der Grafikbibliotheken **220** und **230** ist beim Stand der Technik wohlbekannt. Die 3D-DDI **220** und OPEN GL **230** sind exemplarisch und arbeiten mit einer Anzahl an Computersystemen einschließlich PC-kompatiblen und UNIX-Computern zusammen.

[0041] Gemäß der vorliegenden Erfindung sind die Grafikdarstellungsprozeduren der Grafikbibliotheken **220** und **230** sowie ihre Eingangs- und Ausgangsdatenstrukturen hardwareunabhängig. Diese High-Level-Bibliotheken **220** und **230** sind mit der hardwareabhängigen Low-Level-Grafikbibliothek (HDGL) **240** verbunden, die für die Grafikhardwareeinheit **250** spezifisch ist. Die Grafikhardwareeinheit **250** kann eine Grafikbeschleunigerplatine, eine eingebettete integrierte Schaltung, ein Schaltungsuntersystem innerhalb eines Computersystems für spezielle Zwecke oder ähnliches sein. Eine hardwareunabhängige Kommunikationsschnittstelle **240a** wird zur Bereitstellung der erforderlichen Kommunikationsverknüpfung zwischen den High-Level-Grafikbibliotheken **220**, **230** und der HDGL **240** verwendet. Jede beliebige Anzahl an wohlbekannten Kommunikationsschnittstellen kann für die Schnittstelle **240a** gemäß der vorliegenden Erfindung verwendet werden.

[0042] Im Rahmen der vorliegenden Erfindung kann eine Vielzahl von unterschiedlichen Grafikhardwareeinheiten **250** benutzt werden. Gemäß der vorliegenden Erfindung, lässt sich die HDGL **240** einfach an diese unterschiedliche Hardwareeinheit **250** anpassen, wobei für die hardwareunabhängigen High-Level-Grafikbibliotheken **220** und **230** eine nur geringe oder gar keine Änderung notwendig ist.

[0043] Die HDGL **240** von Fig. 3 beinhaltet einen Satz hardwareabhängiger Low-Level-Prozeduren, die mit den High-Level-Grafikbibliotheken **220** und **230** verbunden sind, welche einen Satz von hardwareunabhängigen Grafikdarstellungsprozeduren aufweisen. Die Schnittstelle verwendet strukturierte, aber hardwareunabhängige Eingangsdatenformate. Das Ausgangsdatenformat der HDGL **240** ist hardwareabhängig und für die Hardwareeinheit **250** spezifisch. Die Prozeduren der HDGL **240** sind auf einem sehr niedrigen Level implementiert (z.B. in der Nähe der Hardwareeinheit **250**) und müssen somit eine nur begrenzte Anzahl an Operationen unterstützen. Da die HDGL **240** eine Low-Level-Bibliothek ist, kann sie zwecks einer Implementierung mit einer

Vielzahl von unterschiedlichen Hardwareeinheiten **250** oder zwecks einer Implementierung mit einer jeweiligen Grafikhardwareeinheit auf einfache Weise umgestaltet werden, anstatt dass (wie in der Vergangenheit) die komplexe Aufgabe einer Umgestaltung der Grafikbibliotheken **220** und **230** mit höherem Level erledigt werden muss. Da die Grafikbibliotheken **220** und **230** hardwareunabhängige Grafikdarstellungsprozeduren gemäß der vorliegenden Erfindung enthalten benötigen diese Bibliotheken kein Umgestalten oder Umschreiben irgendeiner verwendbaren Hardwareeinheit **250**.

[0044] Unter der vorliegenden Erfindung beruhen die High-Level-Grafikbibliotheken **220** und **230** zur Durchführung der für die Bilddarstellung erforderlichen hardwarespezifischen Funktionalität auf der HDGL **240** der vorliegenden Erfindung. Auf diese Weise kann das computergesteuerte Grafiksystem **200** von Fig. 3 leicht an eine Vielzahl von unterschiedlichen Hardwareeinheiten **250** angepasst werden, indem die HDGL **240** mit reduzierter Komplexität (Low-Level-Bibliothek) modifiziert wird, weshalb keine Modifizierung der High-Level-Bibliotheken **220** und **230** notwendig ist. Wahlweise kann eine Anzahl an unterschiedlichen HDGLs **240** in dem System **200** vorgesehen werden, wobei jede unterschiedliche HDGL **240** für eine bestimmte Hardwareeinheit **250** spezifisch ist. In dieser alternativen Ausführungsform ermöglicht die vorliegende Erfindung einem Anwender die Auswahl einer bestimmten zu verwendenden Hardwareeinheit **250**, wobei die geeignete HDGL, die der gewählten Hardwareeinheit **250** entspricht, automatisch von dem System **200** benutzt wird.

[0045] Bei dem Kompilieren und Verlinken der Anwendung **210** werden die erforderlichen Prozeduren der Bibliotheken **220** und **230** sowie die Prozeduren und andere notwendige Elemente der HDGL **240** miteinander verknüpft, um die ausführbare Form der Anwendung **210** zu generieren.

[0046] Fig. 4 ist ein Datenstromdiagramm und illustriert den relevanten Datenstrom zwischen Ebenen des computergesteuerten grafischen Anzeigesystems **200** der vorliegenden Erfindung zum Erzeugen eines Bildes auf dem Bildschirm. Der mit dem Datenstrom assoziierte Verfahrensablauf ist in Fig. 7 illustriert und wird weiter unten separat beschrieben werden.

[0047] Mit Bezug auf Fig. 4 generiert die High-Level-Anwendung **210** eine hardwareunabhängige Grafikdarstellungsanfrage ("Grafikanfrage") einschließlich einer Datenstruktur, die repräsentativ für ein anzuzeigendes Darstellungsgrundelement oder Bild ist. Diese hardwareunabhängige Datenstruktur **410** kann Daten zur Darstellung eines individuellen Darstellungsgrundelements oder andere Grafikdarstellungsbefehle wie z.B. Bitleveltransfers (BLTs) oder Füllungen beinhalten. Die Datenstruktur **410** kann aus einem einzelnen Grundelement oder aus einer Mehrzahl von Grundelementen und/oder Befehlen bestehen. Eine einzelne hardwareunabhängige Datenstruktur **410** kann zu einem Zeitpunkt zu den High-Level-Grafikbibliotheken **220** oder **230** oder viele individuelle Anfragen können gleichzeitig in einem Feldformat zu den Bibliotheken **220** oder **230** übermittelt werden.

[0048] Die High-Level-Grafikbibliothek **220** oder **230** empfängt für jede Grafikanfrage die hardwareunabhängigen Datenstrukturen **410** und sammelt sie, bis eine Gruppe von Datenstrukturen **410** von der Anwendung **210** empfangen wird. Die Größe der Gruppe ist variabel und wird auf der Basis der zulässigen Größe des Stapelverarbeitungsfeldes **420** bestimmt, das durch die High-Level-Grafikbibliotheken **220** oder **230** in Ansprechen auf die Datenstrukturen **410** generiert wird. Ein wie in Fig. 6 dargestelltes Stapelverarbeitungsfeld **420** ist hardwareunabhängig und beinhaltet eine Sequenz von Stapelverarbeitungszellen, z.B. **420a**, **420b**, **420c** usw. Jede Stapelverarbeitungszelle repräsentiert mindestens ein zu generierendes Darstellungsgrundelement und/oder einen auszuführenden Grafikdarstellungsbefehl und enthält einen Operand sowie einen mit dem Operand assoziierten Datensatz. Die Anzahl an Stapelverarbeitungszellen innerhalb eines Stapelverarbeitungsfeldes **420** ist variabel und kann durch den Anwender programmiert werden. Dann wenn eine bestimmte Anzahl an Stapelverarbeitungszellen, z.B. x, bestimmt ist, baut die High-Level-Grafikbibliothek **220** in dem Speicher **102** (Fig. 2) ein bestimmtes Stapelverarbeitungsfeld **420** auf, bis sich x Stapelverarbeitungszellen angesammelt haben oder bis irgend ein anderer zeitkritischer Punkt erreicht wird.

[0049] Bezugnehmend auf Fig. 4 wird das Stapelverarbeitungsfeld **420**, wenn es in dem Speicher (z.B. **102**) aufgebaut ist, zu der hardwareabhängigen HDGL **240** der vorliegenden Erfindung übertragen. Die HDGL **240** verarbeitet die Zellen des Stapelverarbeitungsfeldes **420** sequenziell. Für jede Zelle werden die hardwareunabhängige Datenstruktur und der Operand der Stapelverarbeitungszelle zu hardwareabhängigen Mikrobefehlen umgesetzt, die zu einer hardwareabhängigen Anzeigeliste **430** hinzugefügt werden. Ebenfalls wird die Anzeigeliste **430** ist in einem Speicher (z.B. **102** gespeichert. Die Mikrobefehle in der Anzeigeliste **430** werden von der Hardwareeinheit **250** zur Darstellung der Darstellungsgrundelemente und/oder Grafikdarstellungsbefehle auf dem Bildschirm **105** verwendet. Durch die Verarbeitung von Grafikbefehlen und Grundelementen, die von der Anwendung **210** auf der Basis von Stapelverarbeitungsfeldern erzeugt worden sind, verwenden die

hardwarespezifischen Darstellungsprozeduren der HDGL **240** der vorliegenden Erfindung diejenigen Daten- und Codecache-Ressourcen, die innerhalb des Systems **200** der vorliegenden Erfindung verfügbar sind, auf effiziente Weise.

[0050] Fig. 5 ist ein Logikblockdiagramm und illustriert die Hauptkomponenten der HDGL **240** der vorliegenden Erfindung. Die HDGL **240** beinhaltet einen Satz von Low-Level-Prozeduren (Blöcke **320** und **330**), die in Kombination mit zugeordneten Datenstrukturen (Block **310**) verwendet werden, welche wiederum direkt mit der Hardwareeinheit **250** verbunden sind, indem Mikrobefehle in einer Anzeigeliste im Speicher generiert werden (die in dem System **112** gespeichert oder innerhalb der Hardwareeinheit **250** zugewiesen werden können). Die HDGL **240** empfängt (hardwareunabhängige) Grafikinformationen von den High-Level-Grafikbibliotheken **220** und **230** über einmehrere darzustellende/s graphische/s Bild oder Bilder und generiert aus ihnen eine hardwareabhängige Anzeigeliste, die zu der Hardwareeinheit **250** geführt und zur Darstellung der Bilder auf der Anzeige **105** verwendet wird. Die von der HDGL **240** empfangenen Grafikinformationen zur Erzeugung des graphischen Bilds liegen typischerweise in der Form von definierten Darstellungsgrundelementen und Grafikbefehlen vor. Die von der HDGL **240** generierte Anzeigeliste ist eine Liste mit Mikrobefehlen, die hardwareabhängig sind und von der Hardwareeinheit **250** zur Darstellung der Grundelemente verwendet werden, damit das Grafikbild generiert werden kann.

[0051] Mit Bezug auf Fig. 5 beinhalten die Datenstrukturen **310** der HDGL **240** Eingangsstrukturen zur Aufnahme von Daten der Darstellungsgrundelemente in einem bestimmten hardwareunabhängigen Format gemäß der vorliegenden Erfindung und zur Aufnahme weiterer Operanden und Texture-Maps. Der Block **320** weist einen Satz von Operationen oder "Parametrisierungen" auf, die von der HDGL **240** zur Transformierung der hardwareunabhängigen Grafikbefehle und Grundelemente zu hardwareabhängigen Anzeigelistemikrobefehlen durchgeführt werden. Wie nachstehend erläutert werden wird werden die Eingangsgrafikbefehle und Grundelemente in Stapelverarbeitungszellen gespeichert. Ebenfalls weist der Block **320** Prozeduren zum Einstellen der Darstellungsqualität und Darstellungsleistung des Systems **200** auf. Der Block **330** beinhaltet Texture-Mapping-Prozeduren zum Registrieren (z.B. Übersetzen), Laden und Darstellen von Texture-Maps. Registrierungsprozeduren des Blocks **330** werden zur Umsetzung von Texture-Map-Daten zwischen in eine Farbpalette indizierenden Formaten und RGB-Alpha-Daten verwendenden Formaten benutzt. Eine exemplarische Implementierung der HDGL **240** wird im Abschnitt II dargestellt.

[0052] Fig. 7 ist ein Ablaufdiagramm von Logikblöcken eines Darstellungsverfahrens **500** gemäß der HDGL **240** der vorliegenden Erfindung. Es versteht sich, dass das Verfahren **500** in einem Computersystem wie z.B. einem exemplarischen Computersystem **112** implementiert ist.

[0053] Bei einem Logikblock **510** fragt die High-Level-Anwendung **210** ab, ob ein Darstellungsgrundelement und oder eine Operation oder eine Gruppe von Grundelementen und/oder Grafikdarstellungsoperationen ausgeführt werden soll. Die diese Anfragen repräsentierenden Datenstrukturen **410** werden von der Anwendung **210** zugeführt. Die High-Level-Anwendung **210** kann ein Grundelement zu einem Zeitpunkt abfragen oder die Abfrage kann aus einer Anzahl an individuellen Grundelementen und/oder Grafikdarstellungsoperationen zusammengesetzt sein, die über einen bestimmten Zeitraum hinweg abgefragt werden. Das Format der in dem Block **510** erfolgenden Grafikdarstellungsanfragen ist hardwareunabhängig.

[0054] Bei einem Logikblock **515** empfangen hardwareunabhängige Grafikdarstellungsprozeduren der High-Level-Grafikbibliotheken **220** oder **230** die Grundelemente und/oder Grafikdarstellungsoperationen und konstruieren ein Stapelverarbeitungsfeld **420** auf der Basis einer einzelnen Abfrage oder einer Anzahl an von dem Block **510** sequenziell empfangenen Anfragen. In Abhängigkeit von der zulässigen Größe des Stapelverarbeitungsfeldes **420** können mehrere individuelle Stapelverarbeitungsfelder **420** erforderlich sein, um die von dem Block **510** empfangenen Anfragen zu verarbeiten. Das Stapelverarbeitungsfeld **420** wird in dem Speicher **102** gespeichert und die letzte Stapelverarbeitungszelle des Stapelverarbeitungsfeldes **420** wird als eine "Stapelend"-Zelle gekennzeichnet. Ein Teil des aufgebauten Stapelverarbeitungsfeldes **420** wird in dem Datencache **102b** gespeichert, wie in Fig. 8A dargestellt. Wenn das Stapelverarbeitungsfeld **420** klein genug ist, kann das gesamte Stapelverarbeitungsfeld **420** in den Datencache-Speicher **102b** passen. Obwohl eine Anzahl an unterschiedlichen Speichergrößen auf effektive Weise mit der vorliegenden Erfindung verarbeitet werden kann, liegt eine exemplarische Größe des Datencache-Speichers in der Größenordnung von 8 oder mehr Kilobyte. Das Stapelformat des Stapelverarbeitungsfeldes **420** ist hardwareunabhängig.

[0055] Bei einem Logikblock **520** von Fig. 7 wird das Stapelverarbeitungsfeld **420** zu der HDGL **240** der vorliegenden Erfindung übertragen. Es liegt im Rahmen der vorliegenden Erfindung, dass für den Schritt **520** keine tatsächliche Speicherübertragung erforderlich ist, sondern dass stattdessen ein Zeiger zu der HDGL **240**

übertragen werden kann, der die Ausgangspeicherstelle des Stapelverarbeitungsfeldes **420** in dem gemeinsam genutzten Speicher **102** angibt. Bei einem Logikblock **525** werden die Parametrisierungsroutinen (Block **320** von Fig. 4) der HDGL **240** für eine individuelle Verarbeitung jeder Stapelverarbeitungszelle des Stapelverarbeitungsfeldes **420** verwendet, um hardwarespezifische Mikrobefehle zu erzeugen, die zu einer Anzeigeliste **430** im Computerspeicher (z.B. dem Speicher **102** oder einem anderen Speicher, auf den die Hardwareeinheit **250** direkt zugreifen kann) hinzugefügt werden. Die Parametrisierungsroutinen arbeiten in einer Programmschleife das gesamte Stapelverarbeitungsfeld **420** ununterbrochen ab, bis das "Stapelende" erreicht wird. Die Parametrisierungsroutinen **320** sind so konfiguriert, dass sie vollständig in den Codecache-Speicher **102a** passen wie in Fig. 8A dargestellt.

[0056] Da sich eine Anzahl an Stapelverarbeitungszellen in dem Stapelverarbeitungsfeld **420** in dem Datencache **102b** befindet und da die Parametrisierungsprozeduren **320** der HDGL **240** in dem Codecache **102a** liegen, stellt in Fig. 8A die vorliegende Erfindung einen effizienten Verarbeitungsmechanismus für diese Stapelverarbeitungszellen bereit, weil keine Datencache- oder Codecache-Fehlgriffe für die in Fig. 8A dargestellten Daten auftreten. Mit anderen Worten befinden sich gemäß der vorliegenden Erfindung die meisten Daten und der gesamte Code, der für die Durchführung der Parametrisierung erforderlich ist, in dem Cache-Speicher. Während diese Stapelverarbeitungszellen in dem Cache **102b** bearbeitet werden, wird die Anzeigeliste **430** aufgebaut, indem repräsentative Mikrobefehle für jede Stapelverarbeitungszelle zu der Anzeigeliste **430** mittels der Parametrisierungsprozeduren **320** der HDGL **240** hinzugefügt werden. Wie in Fig. 7 durch den Block **525** illustriert, lädt die vorliegende Erfindung, wenn die Stapelverarbeitungszellen des Speichercaches **102b** vollständig von der HDGL **240** verarbeitet worden sind, eine weitere Gruppe von Stapelverarbeitungszellen in den Datencache **102b**, wie in Fig. 8B dargestellt, und diese Gruppe wird wiederum auf effiziente Weise ohne Daten- oder Codecache-Fehlgriffe von den Parametrisierungsprozeduren **320** verarbeitet. Wie in Fig. 8C gezeigt wird das Verfahren für noch eine weitere Gruppe von Stapelverarbeitungszellen des Stapelverarbeitungsfeldes **420** wiederholt.

[0057] Das Verfahren **525** von Fig. 7 wird solange wiederholt, bis eine Stapelendzelle in dem Stapelverarbeitungsfeld **420** auftritt. Zu diesem Zeitpunkt wird die hardwareabhängige Anzeigeliste **430** als vollständig erachtet. An einem Logikblock **530** wird die Anzeigeliste **430** zu der Hardwareeinheit **250** übertragen, um dargestellt zu werden. Es liegt im Rahmen der vorliegenden Erfindung, dass für den Schritt **530** kein tatsächlicher Speichertransfer erforderlich ist, sondern dass stattdessen ein Zeiger zu der Hardwareeinheit **250** übertragen werden kann, der die Ausgangspeicherstelle der Anzeigeliste **430** in dem gemeinsam benutzten Speicher **102** angibt. Bei dem Block **530** werden die Mikrobefehle der Anzeigeliste von der Hardwareeinheit **240** verarbeitet und ein Bitmap-Bild wird auf dem Bildschirm **105** generiert und so lange in einem Bildspeicher oder einem anderen Videospeicher gehalten, bis es modifiziert oder überschrieben wird. Während die Hardwareeinheit **250** die Verarbeitung der Anzeigeliste **430** durchführt, kann der Prozessor **101** die Befehle der Anwendung **210** verarbeiten.

[0058] Durch die Verarbeitung eines Stapelverarbeitungsfeldes **420** von Darstellungsgrundelementen und/oder Grafikdarstellungsoperationen durch die HDGL **240** der vorliegenden Erfindung werden die Daten- und Cache-Speichereinheiten **102b** und **102a** auf effiziente Weise zum Speichern und Zuführen der erforderlichen Befehle und Daten verwendet, um sequenziell Stapelverarbeitungszellen zu verarbeiten. Unter Verwendung dieses Verfahrens treten während der Parametrisierung (z.B. im Block **525**) nur wenige Datencache-Fehlgriffe und keine Codecache-Fehlgriffe auf.

[0059] Darstellungsqualität/Leistungseinstellung. Die vorliegende Erfindung HDGL **240** ermöglicht dem Anwender weiterhin wählbare Einstellungen, die das Leistungsniveau der von der HDGL **240** ausgeführten Darstellung (z.B. die Geschwindigkeit) und dementsprechend das Niveau der Bildqualitätsdarstellung verändern. Im Einzelnen stellt die HDGL **240** ein in Fig. 9A dargestelltes Leistungs-/Qualitäts-Steuerfeld dar, in dem der Anwender Einstellungen vornehmen kann. Das Leistungs-/Qualitäts-Steuerfeld **610** oder "Wahlfeld" der vorliegenden Erfindung wird auf dem Bildschirm **105** angezeigt. Das Steuerfeld **610** weist einen von einem Anwender einstellbaren Einstellungsanzeiger **615** auf, der durch die Tastatursteuerung über die Einheit **106** bzw. durch einen Cursor **107** oder durch eine ähnliche Bildschirmschnittstelle verändert werden kann. Der Einstellungsanzeiger **615** kann zusammen mit dem Wahlfeld **610** zur Veränderung der Qualitätseinstellung wie von einem Abschnitt **610a** angegeben eingestellt werden, wodurch die Leistungseinstellung entsprechend verändert wird, wie durch einen Abschnitt **610b** angegeben. Das Wahlfeld **610** gibt die Minimal- und Maximaleinstellungen für die Qualität und Leistung vor (wobei in einer Ausführungsform ein Umfang von 0 bis 255, der die dezimalen Bereiche einer 8 bittigen Zahl repräsentiert, verwendet wird). Es können auch andere Wahlfeldformate benutzt werden (z.B. kreisförmig usw.).

[0060] Die Minimal- und Maximaleinstellungen der Qualität und Leistung von Fig. 9A sind in ihrer Reihenfolge umgekehrt, um die inversen Beziehungen zwischen den beiden Charakteristika darzustellen. Gemäß des Wahlfelds **610** wünscht der Anwender bei einer Erhöhung der Darstellungsqualität, dass die Bild- oder Darstellungsqualität verbessert wird. Dieser Vorgang verringert automatisch den Wert der Darstellungsleistung, da das Grafiksystem **200** eine höhere Bearbeitungszeit zur Bewerkstelligung der erwünschten Bildqualität benötigt. Wenn umgekehrt dazu die Darstellungsleistung erhöht wird, verringert das Grafiksystem **200** das Niveau der Bildqualität, um die Grafikinformationen mit höherer Geschwindigkeit durch die Hardwareeinheit **250** verarbeiten zu lassen. Mit Bezug auf eine exemplarische Ausführungsform illustriert der Abschnitt II die Prozedur SetQualityDial, die zur Eingabe des Werts der Einstellung **615** verwendet wird.

[0061] Ein Logikverfahren **620** von Fig. 9B illustriert die Verarbeitung der HDGL **240**, die auf die Einstellungen **615** in dem Leistungs-/Qualitäts-Steuerfeld **610** von Fig. 9A anspricht. Das Verfahren **620** wird unter Verwendung eines Computersystems der Art implementiert, die in Fig. 2 dargestellt ist. Ein Logikblock **625** stellt das Niveau der Perspektivendarstellung von Grafikbildern auf der Basis der Stellung des Einstellungsanzeigers **615** ein. Wie in Fig. 10A dargestellt kann ein Grafikelement **650** auf der Basis eines linearen Modells in Unterteilungen **650a** unterteilt werden. Eine lineare Unterteilung erfordert keine große Berechnungszeit und ermöglicht die Verarbeitung des Systems **200** mit einer hohen Leistungsrate für eine gute Darstellungsleistung. Jedoch und wie in Fig. 10B dargestellt kann ein graphisches Element **655** auch auf der Grundlage eines perspektivischen Modells unterteilt werden, das die dreidimensionale Ausrichtung des Elements auf dem Bildschirm **105** vergleichsweise besser als das lineare Modell illustriert. Fig. 10B stellt dar, dass die Unterteilungen **655a** nicht linear erstellt sind, sondern teilweise auf der dreidimensionalen Ausrichtung des Elements **655** beruhen und auf der Basis der Tiefendimension (z.B. der Z-Achse **657**) des Elements **655** abgestuft sind (um die Perspektive darstellen zu können). Eine perspektivische Unterteilung stellt ein Grafikbild mit höherer Qualität dar, aber sie ist berechnungsintensiv und verringert die Darstellungsleistungsrate.

[0062] Das Verfahren **625** von Fig. 9B reduziert das Ausmaß an perspektivischer Unterteilung und erhöht das Ausmaß an linearer Unterteilung, wenn der Einstellungsanzeiger **615** zur Steigerung der Darstellungsleistungsrate bewegt wird. Ähnlich dazu erhöht das Verfahren **625** das Ausmaß an perspektivischer Unterteilung und verringert das Ausmaß an linearer Unterteilung, wenn der Einstellungsanzeiger **615** zur Steigerung der Darstellungsqualität bewegt wird. Jede beliebige Funktion kann gemäß der vorliegenden Erfindung dazu verwendet werden, um das Ausmaß an perspektivischer/linearer Unterteilung auf der Basis einer vorgegebenen Einstellung **615** zu bewerkstelligen. In einer Ausführungsform wird eine Grenzwertbestimmung ausgeführt, ob die Einstellung **615** jenseits des Mittelwerts in Richtung Leistung (**610b**) liegt, woraufhin die gesamte Unterteilung linear erfolgt, oder ob die Einstellung **615** jenseits des Mittelwerts in Richtung Qualität (**610a**) liegt, wobei die gesamte Unterteilung dann perspektivisch erfolgt. In anderen Ausführungsformen wird bei einer gesteigerten Leistungsrate das Ausmaß oder die Anzahl an linearen Unterteilungen auf Kosten des Ausmaßes an perspektivischen Unterteilungen inkrementell erhöht und bei einer Steigerung der Qualität findet das Gegenteil statt.

[0063] Der Logikblock **630** von Fig. 9B passt die Polygon-(z.B. Dreieck)-Fehlerkorrekturfaktoren auf der Basis der Einstellung **615** an. Wenn sich wie in Fig. 10C dargestellt zwei Dreiecke **660** und **670** in einer Fläche **675** überlappen, wird eine spezielle Prozedur zum Berechnen von Fehlerkorrekturfaktoren bzw. -termen für eine präzise Darstellung der Überlappungsfläche **675** verwendet. Diese Fehlerkorrekturprozedur und die von der vorliegenden Erfindung verwendeten Fehlerkorrekturfaktoren sind in der US-Patentanmeldung mit der Seriennr. 08/299 739, Anwaltsdocketnr. 984128 US, mit dem Titel "Incremental Orthogonal Error Correction for 3D Graphics", eingereicht am 01.09.1994 von Thomas Dye, beschrieben, wobei diese Anmeldung auf den Anmelder der vorliegenden Erfindung übertragen ist. Wenn gemäß des Blocks **630** die Einstellung **615** eine höhere Leistungsrate angibt, verringert bzw. eliminiert die HDGL **240** der vorliegenden Erfindung das Ausmaß an Fehlerkorrekturfaktoren, die von dem obigen Verfahren berechnet und benutzt werden. In dieser Situation werden die Dreiecke **660** und **670** mit einer geringeren Bildqualität, jedoch mit einer guten Darstellungsleistungsrate dargestellt. Alternativ dazu kann, wenn die Einstellung **615** eine höhere Bildqualität angibt, die HDGL **240** der vorliegenden Erfindung das Ausmaß an Fehlerkorrekturfaktoren, die von dem obigen Verfahren berechnet und verwendet werden, erhöhen bzw. maximieren. In dieser Situation werden die Dreiecke **660** und **670** mit einer besseren Bildqualität, aber auch mit einer niedrigeren Leistungsrate dargestellt.

[0064] Ein Logikblock **635** von Fig. 9B steuert die Grenzgrößeneinstellung eines zu einer Abschaltung der Perspektivendarstellung verwendeten Grundelements. Das heißt, die Bildqualität von Grundelementen bestimmter kleiner Größen profitiert aufgrund ihrer reduzierten Größe nicht wesentlich von den Perspektivendarstellungstechniken. Eine Grenzgröße wird von der HDGL **240** aufrechterhalten, die das perspektivische Rendering für sämtliche Grundelemente unterhalb der Grenzgröße abschaltet. Wird die Leistungs-/Qualitäts-Ein-

stellung **615** (Fig. 9A) hinsichtlich einer höheren Bildqualität eingestellt, wird die von dem Block **635** der vorliegenden Erfindung aufrechterhaltene Grenzgröße verringert, sodass die meisten Grundelemente unter Verwendung der Perspektivendarstellungstechniken dargestellt werden. Wenn die Leistungs-/Qualitäts-Einstellung **615** (Fig. 9A) für eine höhere Darstellungsleistungsrate eingestellt ist, wird die von dem Block **635** der vorliegenden Erfindung aufrechterhaltene Grenzgröße erhöht, sodass für eine höhere Leistung zunehmend größere Darstellungsgrundelemente oder Bilder nicht unter Verwendung der Perspektivendarstellungstechniken angezeigt werden. Es können jede beliebigen Funktionen gemäß der vorliegenden Erfindung benutzt werden, um eine Grenzwert-Abschaltgröße auf der Grundlage einer vorgegebenen Einstellung **615** zu bewerkstelligen.

[0065] Texture-Map-Formatumsetzungen. Die HDGL **240** der vorliegenden Erfindung stellt ebenfalls eine Umsetzung zwischen Texture-Map-Formaten bereit, um Texture-Informationen (eine "Textur") zu registrieren. In einer exemplarischen Ausführungsform werden zwei Texture-Formate in der in dem Computersystem implementierten Umsetzungsprozedur **700** von Fig. 11 der vorliegenden Erfindung verwendet. Ein Format ist das RGB-Alpha-Format, wobei jedem Pixel eine Rot-, Grün-, Blau- und ein Alpha-Wert zugewiesen wird. Ein zweites Format ist ein Indexformat, bei dem jedem Pixel ein Indexwert in einer Farbpalette zugewiesen wird. Das Vorgabeformat hängt von dem Format ab, auf das die Hardwareeinheit **250** angepasst ist. Jedes der obigen Formate oder auch jedes andere Format kann das Vorgabeformat sein. Die Prozedur **700** illustriert eine exemplarische Ausführungsform, bei der das RGB-Alpha-Format das Vorgabeformat ist.

[0066] Gemäß des Verfahrens **700** empfängt die HDGL **240** an einem Logikblock **705** ursprüngliche Texture-Informationen in einem bestimmten Texture-Format. Diese ursprünglichen Texture-Informationen oder -Daten werden in einem Speicher **102** angeordnet. An einem Logikblock **710** überprüft die vorliegende Erfindung ein vorbestimmtes Flag, um zu bestimmen, ob die Texture-Informationen in dem Indexformat vorliegen (z.B. Indizes in einer bestimmten Farbpalette). Wenn ja fährt die Verarbeitung mit einem Logikblock **715** fort, wo die Texture-Daten von dem Indexformat in ein RGB-Alpha-Format umgesetzt werden. An dem Block **715** wird für jedes Pixel der Texture-Daten der Indexwert zum Erhalt der jeweiligen Farbattribute aus einer Farbpalette für dieses Pixel verwendet. Ist das Farbattribut des Pixels ermittelt worden, werden seine Rot-, Grün-, Blau-, und Alpha-Werte bestimmt und in einem RGB-Alpha-Format aufgezeichnet. Anschließend werden die sich ergebenden Texture-Informationen an einer neuen Stelle oder an der gleichen Stelle in dem Speicher **102** der ursprünglichen Texture-Informationen abgespeichert (wobei in letzterem Fall die ursprünglichen Texture-Informationen z.B. überschrieben werden). Dann fährt die Verarbeitung mit einem Block **720** fort.

[0067] Wenn an dem Block **710** das Indexformat nicht im Indexmodus vorliegt, wird davon ausgegangen, dass das Format RGB-Alpha ist. Zu dem Zeitpunkt, wenn die Verarbeitung mit dem Block **720** fortfährt, wird die Textur als registriert betrachtet. Dann geht die Verarbeitung zu dem Logikblock **720** über, wo die Texture-Informationen übertragen und für eine Verwendung durch die Parametrisierungsprozeduren **320** der HDGL **240** zugewiesen werden. In Abhängigkeit von der Hardwareeinheit **250** beteiligt dies das Laden der Textur von dem Speicher **102** (Systemspeicher) zu einem Speicher **102'**, auf den die Hardwareeinheit **250** direkt zugreifen kann (wobei dieser Speicher typischerweise in der Hardwareeinheit liegt). Dann können die registrierten Texture-Informationen mittels einer Anzahl an Polygondarstellungsoperationen angezeigt werden, die von der vorliegenden Erfindung berücksichtigt und in den im Abschnitt II beschriebenen Ausführungsformen illustriert sind.

[0068] Es versteht sich, dass die in Fig. 11 dargestellten Umsetzungsverfahren **700** für jede beliebigen Grafikinformationen neben Texture-Informationen für eine Anzeige durchgeführt werden können. Beispielsweise können die den Grundelementen zugeordneten Scheitel in sowohl den Index- wie den RGB-Alpha-Formaten spezifiziert werden und die vorliegende Erfindung sorgt in Abhängigkeit von der vorgegebenen Voreinstellung für eine Umsetzung zwischen den beiden Formaten (siehe den nachstehenden Abschnitt II für die jeweiligen Beschreibungen).

[0069] Parametrisierung. Fig. 12 illustriert ein Ablaufdiagramm einer computerimplementierten Logikprozedur **800** der HDGL **240**, die zur Abfrage einer Stapelverarbeitungszelle des Stapelverarbeitungsfeldes **420** und zur Durchführung einer Parametrisierung daran verwendet wird, um Mikrobefehle für die Anzeigeliste **430** zu erstellen. Das Verfahren **800** entspricht dem Verfahren **525** von Fig. 7. Auf eine bestimmte Implementierung der Prozedur **800** wird in dem nachstehenden Abschnitt II als BuildDisplayList Bezug genommen. Das Verfahren **800** beginnt bei einem Logikblock **805**, in dem die vorliegende Erfindung überprüft, ob die Grafikprozeduren initialisiert sind (z.B. wird überprüft, ob InitGraph ausgeführt wurde), und wenn nicht, wird die Verarbeitung über einen Logikblock **855** beendet. Wenn die Grafik initialisiert ist, geht die Verarbeitung zu einem Logikblock **810** über, wo eine Stapelverarbeitungszelle des Stapelverarbeitungsfeldes **420** erhalten und die Parameter von der Stapelverarbeitungszelle abgefragt werden. In eine Ausführungsform (die nachstehend in Abschnitt II darge-

stellt ist), enthält jede Stapelverarbeitungszelle ein Operandenfeld, ein Zahlenfeld (das eine mit den Daten in der Zelle assoziierte Anzahl an Scheiteln angibt), ein Flagfeld sowie Datenstrukturen (oder ein darauf weisender Zeiger) für jeden Scheitel (z.B. Farbattribute, Koordinaten usw.). Das Operandenfeld definiert ein darzustellendes Grundelement oder eine/n auszuführende/n Grafikoperation bzw. -befehl.

[0070] In Abhängigkeit von dem Operand der Zelle wird eine Prozedur von **820**–**845** von dem Umschaltlogikblock **815** aufgerufen. Das Verfahren des Auslesens der Parameter der Stapelverarbeitungszelle und der daraus generierten Mikrobefehle zur Anordnung in der Anzeigeliste **430** wird als "Parametrisierung" bezeichnet und von den Logikblöcken **820**–**845** ausgeführt. Wenn der Operand einen Bitleveltransfer ("BLT") anzeigt, wird ein BLT durch den Logikblock **820** zwischen einer ersten Bildschirmfläche und einer zweiten Bildschirmfläche ausgeführt. Die erste Bildschirmfläche wird durch zwei Eckenkoordinaten (erster und zweiter Scheitel) und die zweite Bildschirmfläche wird durch eine Koordinate (dritter Scheitel) identifiziert. Anschließend werden die Pixel innerhalb der ersten Koordinate durch die BLT-Operation in die zweite Koordinate kopiert. Auf der Basis des Flagfeldes können die Texture-Informationen als die Transferquelle verwendet werden. Weiterhin synchronisiert die Einstellung eines zweiten Flags den Transfer mit einer Bildschirmaktualisierung. Auf der Grundlage der BLT-Operation generiert die HDGL **240** der vorliegenden Erfindung die geeigneten Mikrobefehle innerhalb der hardwareabhängigen Anzeigeliste **430**.

[0071] Wenn der Operand von dem Block **815** ein FILL-Operand ist, wird der Logikblock **825** ausgeführt, wobei eine von zwei Ecken (erster und zweiter Scheitel) spezifizierte Bildschirmfläche mit einer spezifischen Quelle von Informationen aufgefüllt wird. Die für die Füllung ausgewählte Farbe stammt von dem Farbattribut, das in der ersten Scheitel-Datenstruktur eingestellt ist. Wenn in dem Stapelflag eine Z-Pufferung eingestellt ist, wird der Z-Puffer mit einem in dem Zahlenfeld definierten Wert gefüllt. Auf der Grundlage der FILL-Operation generiert die HDGL **240** der vorliegenden Erfindung die geeigneten Mikrobefehle innerhalb der hardwareabhängigen Anzeigeliste **430**.

[0072] Wenn der Operand von dem Block **815** von Fig. 12 ein POINT-Operand ist, wird der Logikblock **830** zur Generierung von Mikrobefehlen in der Anzeigeliste **430** ausgeführt, um eine Anzahl an Punkten anzuzeigen, die in den Scheitel-Datenstrukturen definiert sind, während die Anzahl an zu verarbeitenden Punkten in dem Zahlenfeld angegeben ist. Auf der Grundlage der POINT-Operation generiert die HDGL **240** der vorliegenden Erfindung die geeigneten Mikrobefehle innerhalb der hardwareabhängigen Anzeigeliste **430**.

[0073] Wenn der Operand von dem Block **815** ein POLYLINE-Operand ist, wird der Logikblock **835** zur Generierung von Mikrobefehlen in der Anzeigeliste **430** ausgeführt, um die von der Stapelverarbeitungszelle definierte Linie oder Reihe von Linien anzuzeigen. Die Punkte, welche die Linie bzw. Linien bilden, werden durch die Scheitel-Datenstrukturen festgelegt und die Anzahl an Punkten wird in dem Zahlenfeld definiert. Es werden drei Liniendarstellungsmodi unterstützt, namentlich der Listen-, der Zerlegungs- und der Fächermodus ('list, strip and fan mode'), wobei der Modus in dem Flagfeld gesetzt wird. In dem Listenmodus wird ein Satz unabhängiger Linien dargestellt, wobei jedes Scheitelpaar die Endpunkte jeder Linie repräsentiert. In dem Zerlegungsmodus wird die zweite Linie an den Endpunkt der vorhergehenden Linie angehängt. Die erste Linie wird durch ein Scheitelpaar definiert und jede Linie danach wird durch einen einzelnen Scheitel festgelegt. In dem Fächermodus definiert der erste Scheitel einen Punkt, der für alle Linien verwendet wird (ein gemeinsamer Punkt) und jeder Scheitel danach definiert eine Linie von dem ersten Scheitel zu dem jeweiligen Punkt. In jedem der obigen Modi spezifiziert das Zahlenfeld die Anzahl an Scheiteln in der POLYLINE-Operation. Auf der Grundlage der POLYLINE-Operation generiert die HDGL **240** der vorliegenden Erfindung die geeigneten Mikrobefehle innerhalb der hardwareabhängigen Anzeigeliste **430** für die verarbeitete Stapelverarbeitungszelle.

[0074] Wenn der Operand von dem Block **815** ein POLYGON-Operand ist, wird der Logikblock **840** zur Generierung von Mikrobefehlen in der Anzeigeliste **430** ausgeführt, um die durch die Stapelverarbeitungszelle definierten Polygon-Grundelemente oder Polygonreihen anzuzeigen. Die Anzahl an in der Stapelverarbeitungszelle spezifizierten Scheiteln wird in dem Zahlenfeld angegeben. Es werden drei Polygondarstellungsmodi unterstützt, namentlich der Listen-, der Zerlegungs- und der Fächermodus, wobei der Modus in dem Flagfeld gesetzt wird. In dem Listenmodus legen jeweils drei Scheitel ein Polygon fest. In dem Zerlegungsmodus definieren die ersten drei Scheitel ein erstes Polygon und danach definiert jeder Scheitel ein neues Polygon, das sich mit dem vorhergehenden Polygon zwei Scheitel teilt. In dem Fächermodus wird der erste Scheitel in der Scheitel-Datenstruktur von jedem Polygon geteilt und durch jeweils zwei danach auftretende Scheitel wird ein neues Polygon definiert. Auf der Grundlage der POLYGON-Operation generiert die HDGL **240** der vorliegenden Erfindung die geeigneten Mikrobefehle innerhalb der hardwareabhängigen Anzeigeliste **430** für die verarbeitete Stapelverarbeitungszelle.

[0075] Wenn der Operand von dem Block **815** ein Steueroperand ist, wird der Logikblock **845** zur Verarbeitung des Steueroperanden ausgeführt. Zum Beispiel stellt der Steueroperand SaturatetoBounds bestimmte Farbsteuerregister entsprechend den Werten des ersten und zweiten Bytes in dem Zahlenfeld für die Farbmaskierung und -sättigung auf hoch und niedrig. Der SetZMask-Operand setzt die Z-Maske für ie Kollisionserfassung und Objektidentifizierung auf den Wert des Zahlenfelds. Der SetDisplayPage-Steueroperand stellt den Anfangsversatz des Anzeigebereichs ein. Dies wird für eine Doppel- und Dreifach-Pufferung verwendet. Der Versatzwert wird dem Zahlenfeld zugeführt. Wie im Abschnitt II beschrieben sind auch andere Steueroperanden zulässig.

[0076] Nach der Vervollständigung der Logikblöcke **820**–**845** vollzieht sich die Rückkehr zu einem Block **850**, wo das Flagfeld der verarbeiteten Stapelverarbeitung zur Bestimmung überprüft wird, ob es ein Batch End-('Stapelende')-Flag enthält. Wenn ja, ist das Stapelverarbeitungsfeld **420** vollständig und die Ausführung wird über einen Block **855** verlassen. Wenn das Batch End-Flag nicht gesetzt ist, kehrt die Verarbeitung zu dem Block **810** zurück, um die nächste Stapelverarbeitungszelle des Stapelverarbeitungsfeldes **420** zu holen und zu verarbeiten. Die durch die Verarbeitung von Fig. 12 generierte Anzeigeliste wird an der Anzeigeeinheit **105** dargestellt, wenn das System **200** einen Löschanzeigeliste-Befehl empfängt (siehe Abschnitt II für eine exemplarische Ausführungsform).

[0077] Der für das Verfahren **800** erforderliche Befehlssatz ist ausreichend kompakt, um in die meisten Codecache-Speicher **102a** der meisten Computersysteme zu passen. Auf diese Weise führt die Ausführung eines Stapelverarbeitungsfeldes **420** von Stapelverarbeitungszellen durch das Verfahren **800** nicht zu Codecache-Fehlgriffen und das Parametrisierungsverfahren **800** wird rasch ausgeführt. Durch ein sequenzielles Ausführen der Stapelverarbeitungszellen innerhalb eines Stapelverarbeitungsfeldes **420** ohne Unterbrechung wird die Verwendung des Datencaches **102b** während der Parametrisierung ebenfalls maximiert.

Abschnitt II

[0078] Ein Zweck der HDGL **240** besteht in der Bereitstellung einer einfach anzupassenden hardwareabhängigen Schnittstelle für die hardwareunabhängigen Grafikbibliotheken **220** und **230**. Es ist zu erwarten, dass verschiedene Low-Level-Versionen der HDGL **240** entwickelt werden können, um mit einer Vielzahl unterschiedlicher Hardwareeinheiten betrieben zu werden. Wahlweise kann eine Gruppe von HDGLs mit einer Anzahl an Hardwareeinheiten versehen werden, wobei der Anwender zwischen der Gruppe auswählen kann. Eine exemplarische Implementierung der Version einer HDGL **240** ist nachstehend dargestellt. Obgleich eine Anzahl an alternativen Ausführungsformen innerhalb des Rahmens der HDGL **240** der vorliegenden Erfindung realisiert werden kann, ist die nachstehend aufgeführte exemplarische Implementierung unter der Computersprache C modelliert worden. Im Einzelnen sind exemplarische Implementierungen für den Datenstrukturblock **310** und den Operationsblock **320** der HDGL **240** von Fig. 4 angegeben. Der Fachmann kann spezifische Strukturen und Prozeduren mit der HDGL **240** verwenden, die zu dem erwünschten Ergebnis führen.

[0079] Exemplarische hardwareabhängige Grafikbibliothek-Datenstruktur **310** Die folgenden grundlegenden Feldtypen sind folgendermaßen definiert:

int	32 Bit-Ganzzahl mit Vorzeichen
DWORD	32 Bit-Ganzzahl ohne Vorzeichen
WORD	16 Bit-Ganzzahl ohne Vorzeichen
BYTE	8 Bit-Buchstabe ohne Vorzeichen
FIX	32 Bit-Zahl mit fester Kommastelle, wobei 16 Bits die Ganzzahl und 16 Bits der Bruchteil sind.
BOOL	Ganzzahl, wahr (1)/falsch (0)

[0080] Die HDGL **240** fungiert als eine Parametrisierungslage, wobei die High-Level-Beschreibung **420** der Darstellungsgrundelemente (Punkte, Linien usw.) verarbeitet und der Hardwarelevel-Code generiert und in einer Anzeigeliste **430** gespeichert wird. Das als Eingang in die HDGL dienende Stapelverarbeitungsfeld **420** ist als ein Feld aus Stapelverarbeitungsstrukturen definiert. Jede Stapelverarbeitungszelle enthält mindestens eine Stapelverarbeitungsstruktur. Eine exemplarische Stapelverarbeitungsstruktur (z.B. für eine Stapelverarbeitungszelle des Stapelverarbeitungsfeldes **420**) ist nachstehend angeführt.

Stapel

```
DWORD       dwOp,
DWORD       dwFlags,
DWORD       dwN,
WORD        wTexID,
Vert*       pVert;
```

[0081] Die obige Stapelverarbeitungsstruktur ist ein Prototyp für eine Stapelverarbeitungszelle. Der Wert "dwN" definiert die Anzahl an Punkten der Scheitel in dem der Stapelverarbeitungsstruktur zugeordneten Grundelement. "dwOp" bestimmt die von der Stapelverarbeitungsstruktur ausgeführte grundlegende Grafikoperation und kann eine der folgenden Operationen sein:

1. 2D-Operationen: Der BLT-Operand kopiert eine durch die ersten zwei Scheitel bestimmte rechteckige Fläche, wobei der erste Scheitel z.B. die obere linke Ecke und der zweite Scheitel die untere rechte Ecke definiert (einschließend). Die Bestimmung für die Fläche wird durch den dritten Scheitel in einer Stapelverarbeitungsstruktur eines Stapelverarbeitungsfeldes durch seine obere linke Koordinate definiert. Wenn ein Flag TIMED_BLT gesetzt ist, wird die ausgeführte BLT-Operation durch das Aktualisieren der Videobilder der in dem dwN-Feld der Zelle gespeicherten Liniennummer ausgelöst. Wenn TEXTURE in den Flags vorhanden ist, wird die Texture-Zahl dwN zu der Quelle der BLT-Operation. Der Operand FILL stellt die Farb- oder anderen Attribute einer rechteckigen Fläche des Bildschirms ein. Die Fläche ist durch die ersten beiden Scheitel festgelegt; so definiert z.B. der erste Scheitel die obere linke Ecke und der zweite Scheitel definiert die untere rechte Ecke (einschließend). Wenn eine Z-Pufferung in den dwFlags gesetzt ist, wird der Z-Puffer mit dem Wert von dem dwN-Feld (z.B. die unteren 16 Bits) aufgefüllt werden. In einer Implementierung ist dies normalerweise auf 0xFFFF gesetzt. Die Farbwerte, die den Bildpuffer auffüllen (z.B. eine Speichereinheit, die gemeinsam mit der Hardwareeinheit **250** genutzt wird oder auf die Einheit zugreifen kann) werden den ersten Scheitel-Farbfeldern entnommen. Der FILL-Operand erkennt Füllungen mit einem Wert von Null und stellt je nach Unterstützung von der Hardwareeinheit **250** die Schnell-Lösch-Operation ein.
2. 3D-Operationen: Der POINT-Operand stellt einen Satz an Punkten dar, auf deren Koordinaten durch ein Scheitel-Feld gezeigt wird. Die Anzahl an Punkten wird in dem dwN-Feld definiert. Der POLYLINE-Operand stellt ein mehrliniges Grundelement dar, das aus bis zu (dwN-1) Scheiteln besteht. Der POLYGON-Operand stellt ein Polygon-Grundelement auf der Grundlage der Anzahl dwN der Scheitel dar. Der genaue Polygontyp wird durch ein POLY*-Flag spezifiziert.
3. Grafikdarstellungssteueroperationen: Der SATURATE_TO_BOUNDS-Operand stellt die Farbvergleichsregister in einer Ausführungsform entsprechend den Werten des ersten (1sb) und des zweiten Bytes in 'dwN' im Bereich von 0 bis 255 auf hoch oder niedrig: Dies wird bei Farbmarkierungs- und -sättigungsprozeduren verwendet. Der SET_Z_MASK-Operand setzt die Z-Maske für die Kollisionserfassung und Objektidentifikation auf den Wert (z.B. die unteren 16 Bits) von dwN. Das grundlegende Prinzip einer Objektidentifzierung besteht darin, verschiedene obere Bits der Z-Koordinate zum Aufbewahren der Objektidentifikationsnummer verfügbar zu halten. In einer Ausführungsform ist diese Partition hardwaregeschützt, indem ihre Bits auf Null gestellt werden, während der Rest auf 1 gesetzt wird. Später wird während der Kollisionserfassung der kollidierte Z-Wert betreffs dieser Bit-Zeichenkette untersucht, um die Identifikation des kollidierten Objekts darzustellen. Der SET_DISPLAY_PAGE-Operand setzt den Beginn des Versatzes des Objekts, was für eine Doppel- und Dreifach-Pufferung verwendet wird. Der Versatz wird dem dwN-Feld zugeführt.

[0082] Hinsichtlich der Stapelverarbeitungsstruktur gibt das 'dwFlags'-Feld zusätzliche Merkmale an: Das Batch End-Flag wird in der letzten Zelle in einem Stapelverarbeitungsfeld **420** gesetzt, um die Stapelverarbeitung zu beenden. Wenn nur eine einzige Stapelverarbeitungszelle vorliegt, wird das Stapelende in den 'dwFlags' der Zelle gesetzt. Das TIMED_BLT-Flag wird innerhalb der BLT-Operation gesetzt, um eine getaktete BLT-Operation auszuführen, die mit der spezifizierten Bildschirm-Refreshline synchronisiert wird.

[0083] Das 'dwFlags'-Feld gibt ebenfalls das jeweilige mehrlinige Grundelement und den zu verwendenden Darstellungsmodus der Polygon-Grundelemente an. POLYLINE-Grundelemente können mehrere Linien aufweisen und als Listen, Zerlegungen oder Fächer definiert sein, wobei (1) LINE_LIST, oder (2) LINE_STRIP, oder (3) LINE_FAN spezifiziert wird. Die Polygon-Grundelemente können aus mehreren Dreiecken bestehen und als Listen, Zerlegungen oder Fächer definiert werden, wobei (1) POLY_LIST, oder (2)POLY_STRIP, oder (3) POLY_FAN spezifiziert wird. Die Unterschiede zwischen diesen Darstellungsmodi werden nachstehend beschrieben werden. LIST ist ein geordneter Satz von Punkten, der unter Verwendung von drei Punkten Polygone definiert (z.B. dreieckige Polygone). Das erste dreieckige Polygon wird durch die Punkte; 0, 1, 2 definiert.

Das zweite dreieckige Polygon wird durch die Punkte 3, 4, 5 definiert. Das dritte Polygon wird durch die Punkte: 6, 7, 8 usw. definiert. Ein STRIP ist ein geordneter Satz von Punkten, der die Punkte 0, 1, 2 als das erste dreieckige Polygon benutzt. Das zweite dreieckige Polygon wird durch die Punkte 1, 2, 3 definiert. Das dritte dreieckige Polygon wird durch die Punkte 2, 3, 4 usw. definiert. Ein FAN ist ein geordneter Satz an Punkten, der den Punkt 0 als den Mittelpunkt vieler dreieckiger Polygone verwendet. Das erste dreieckige Polygon wird als 0, 1, 2 definiert. Das zweite dreieckige Polygon wird durch die Punkte 0, 2, 3 definiert. Das dritte dreieckige Polygon wird durch die Punkte 0, 3, 4 usw. definiert.

[0084] Wenn die Operation Z-gepuffert werden soll, wird eines der folgenden Flags in 'dwFLAGS' eingeschlossen: (1) ZBUFFER führt eine normale Z-Operation aus, oder ZALWAYS schreibt beide Pixel und Z; oder ZMASK führt eine normale Z-Operation aus, aber aktualisiert den Z-Puffer nicht. Zusätzlich wird das GOURAUD-Flag zu den Polyline- und Polygonoperationen hinzugefügt, wenn eine Schattierung erwünscht ist. Das ALPHA-Flag wird in bestimmten Hardwareeinheiten, die dieses Merkmal unterstützen, zu Polygonoperationen für ein Alpha-Blending hinzugefügt.

[0085] Das TEXTURE-Flag erzeugt Texture-Maps von Polygonen. Solche texturierten Polygone können ebenfalls PERSPECTIVE aufweisen, das zur Anschaltung der Perspektivenkorrektur hinzugefügt ist. Wie nachstehend erläutert stellt der Wert 'wTexID' die Texture-Identifikationsnummer der registrierten Textur dar.

[0086] Die folgenden Flags werden bei Texturen verwendet: (1) TEX_MAX_NEAREST, eine allgemeine Voreinstellung; oder (2) TEX_MAX_LINEAR; und eines der folgenden Flags: TEX_MIN_NEAREST, oder TEX_MIN_LINEAR, eine weitere allgemeine Voreinstellung. Das Flag TEX_TRANSP wird für transparente Texturen hinzugefügt. Die als transparent behandelte Farbe wird durch die Operation SATURATE_TO_BOUNDS sowohl bezüglich des oberen wie des unteren Werts eingestellt.

[0087] Eine weitere Datenstruktur der HDGL **240** ist die Vert-Struktur, die für ein Darstellungsgrundelement einen Scheitel festlegt:

Vert

```
FIX x, y, z;
union c
    BYTE index;
    BYTE r,g,b,a;
FIX u, v, w;
```

[0088] Die Position im dreidimensionalen Raum des Scheitels wird durch x-, y- und z-Feldkoordinaten festgelegt. Die Farbe kann aus zwei unterschiedlichen Formaten erkannt werden, namentlich dem "Index"-Format (in einer Ausführungsform für den indizierten 8 bpp-Farbmodus) oder einer Kombination von r, g, b, wobei das Alpha-Format als Komponenten spezifiziert ist (in einer Ausführungsform für den 16, 24 bpp-Farbmodus). Die Felder u und v definieren die Koordinate in dem Texture-Raum und das Feld w ist die homogene Koordinate für Texture-Berechnungen. Für jedes Darstellungsgrundelement in einer Stapelverarbeitungszelle wird ein Feld dieser Vert-Strukturen in dem Speicher **102** generiert und die Adresse des Vert-Feldes wird über den 'pVert'-Eintritt zu der HDGL **240** in eine Stapelverarbeitungszelle des Stapelverarbeitungsfeldes **420** übergeleitet. Ebenfalls erleichtert die Struktur die Texture-Map-Koordinaten (u, v), die mittels Eins-zu-Eins-Mapping zu den gegebenen Bildschirmkoordinaten übersetzt werden. Diese werden in Texture-Mapping-Operationen verwendet. Der Parameter "w" ist der perspektivische Faktor dieses Scheitels, dessen Wert auf "1" gesetzt ist, d.h. dass in einer Ausführungsform dem Scheitel keine Perspektive zugeordnet wird, wobei jeder höhere Wert bedeutet, dass die Textur zu dem Scheitel hin abgeschrägter ist, wodurch eine höhere perspektivische Verzerrung bewirkt wird.

[0089] Eine weitere Datenstruktur der HDGL **240** ist die Texture-Struktur, die für Registrierungszwecke ein Texture-Map definiert.

Texture

WORD	wHeapID;
WORD	wWidth;
WORD	wHeight;
BYTE*	pbTex
DWORD	dwFlags

[0090] Diese Datenstruktur verfügt über Texture-Dimensionsfelder (Breite und Höhe) und einen Zeiger (pbTex) zu einer Textur, die in dem Systemspeicher **102** gespeichert ist. Ebenfalls beinhaltet sie das Handle (HeapID) zu dem Heap, der unter Verwendung der Funktion TextureHeapAlloc() sämtlichen Texturen eines einzelnen Anwenders zugewiesen wurde. Das 'dwFlags'-Feld erfasst den Texture-Typ und es kann in Abhängigkeit von der Hardwareeinheit **250** eine der folgenden Einstellungen aufweisen: (1) TEX_4BBP für indizierte 4 bpp-Texturen, oder (2) TEX_8BBP für indizierte 8 bpp-Texturen, oder (3) TEX_16BBP für 565 True-Color-Texturen, oder (4) TEX_24BBP für 888 True-Color-Texturen oder jede beliebige andere anwenderspezifische Einstellung.

[0091] Ebenfalls kann das folgende Flag logisch zu den 'dwFlags' hinzugefügt werden: TEX_PROTECT schützt die Anwender-Textur davor, während der Texture-Erfassung mit ihrem Hardwareformat überschrieben zu werden. Für True-Texturen können die Texture-Daten in einer Ausführungsform in einem 3D-gepackten Format [alpha-BGR] geschrieben werden, wobei die Farbe Rot in dem am geringsten signifikanten Byte linear von links nach rechts und von der obersten zu der untersten Linie gespeichert wird. Für indiziertes 8 bpp ist nur ein Byte pro Pixel erforderlich und für 4 bpp werden zwei Pixel in jedes Byte gepackt, wobei das Pixel auf der linken Seite des Paars in dem oberen Nibble gespeichert wird.

[0092] Eine weitere Datenstruktur der HDGL **240** ist die Palettenstruktur, die einen Palettenzellen-Satztyp festlegt:

Palette (256)

BYTE	r;
BYTE	g;
BYTE	b:

[0093] In einer Ausführungsform definiert diese Struktur einen Typ für einen Satz von 256 Palettenzellen mittels ihrer Rot-, Grün- und Blau-Komponenten.

[0094] Eine weitere Datenstruktur der HDGL **240** ist die Rect-Struktur, die eine einschließende rechteckige Fläche auf dem Bildschirm definiert:

Rect

WORD	x1;
WORD	y1;
WORD	x2;
WORD	y2;

[0095] In einer Ausführungsform definiert das Format (x1, y1) die obere linke Ecke und (x2, y2) definiert die untere rechte Ecke. Eine weitere Datenstruktur der HDGL **240** ist die Init-Struktur:

Init

Word	Auflösung
WORD	color mode;
WORD	texspace;

[0096] Diese Struktur bestimmt die zu initialisierenden Auflösungs- und Farbmodi. Ebenfalls stellt sie die Menge an Texture-(privatem) Speicher ein, der zur Abspeicherung geladener Texturen verwendet wird.

[0097] Eine weitere Datenstruktur der HDGL **240** ist die DisplayContext-Struktur, die den Status der Hardwareeinheit **250** zu jedem Zeitpunkt wiedergibt:

DisplayContext

```
WORD    wHardware;
WORD    wVideoMemory;
WORD    wTextureHeap;
WORD    wTextureAvail;
DWORD   fCapAlpha;
DWORD   fCapTexture;
DWORD   fCapZmask;
```

[0098] Die Adresse dieser systemweiten Struktur wird durch den Aufruf einer anderen Initialisierungsprozedur erhalten. Das wHardware-Feld enthält den Code der darunter liegenden Grafikhardware und gibt die unterstützte Hardware-Version wieder: (1) HARDWARE_A, HARDWARE_B, HARDWARE_C usw.

[0099] Das wVideoMemory-Feld gibt die Menge an Video-RAM an (einschließlich des Z-Puffers), die in dem Hardware-Board vorhanden ist. Das wTextureHeap-Feld zeigt die gesamte für Texturen verfügbare Speichermenge und das wTextureAvail-Feld gibt die für die Zuweisung von Texturen verfügbare Speichermenge an. Dieser Wert ist gleich zu denjenigen von wTextureHeap oder liegt darunter, da unterschiedliche Anwender über bereits zugewiesene Heaps für ihre Texturen verfügen können.

Exemplarische hardwareabhängige Grafikbibliothek-Operationen **320**

[0100] Ein Boolescher Wert wird von einigen Funktionen zurückgegeben, um zu signalisieren, ob die Funktion erfolgreich verlaufen (TRUE) oder fehlgeschlagen (FALSE) ist. In dem Fall eines Fehlschlages kann GetErrCode() aufgerufen werden, um den Fehlercode des letzten Fehlers wiederzugeben und GetErrMsg() kann dazu aufgerufen werden, den Zeiger auf eine mit Null abschließende Zeichenfolge zurückzusetzen, die den letzten Fehler beschreibt.

[0101] Eine HDGL 240-Prozedur InitLib() wird vor jeder anderen Prozedur aufgerufen, um die HDGL **240** zu initialisieren. Eine weitere Prozedur ist die InitGraph-Prozedur:
BOOL InitGraph (const WORD wResolution, const WORD wColorMode)

[0102] Diese Prozedur initialisiert die Grafikhardwareeinheit **250**. Sie wird von dem System **200** abgerufen und spezifiziert die zu initialisierenden Auflösungs- und Farbmodi. Das Feld 'wResolution' ist eines der folgenden Felder: (1) RES GAME setzt die Auflösung auf 640 × 480; (2) RES_STANDARD setzt die Auflösung auf 1024 × 768. Das 'wColorMode'-Feld stellt den erwünschten Modus in dem Grafikmodus entweder auf: (1) COL8 für 8 bpp palletiert, auf (2) COL 16 für 16 bpp 5-6-5, oder auf (3) COL_24 für 24 bpp True-Color-8-8-8.

[0103] In Abhängigkeit von der Hardwareeinheit **250** lauten valide Kombinationen von 'wResolution' und 'wColorMode' wie folgt: (1) 640 × 480 × 8 indiziert, db mit Y-Versatz 640 × 480 × 8 indiziert, db mit Anzeigezeigerumschaltung; (2) 640 × 480 × 16 tc, db mit Y-Versatz; (3) 640 × 480 × 24 tc, db mit Y-Versatz; (4)1024 × 768 × 8 indiziert; (5) 1024 × 768 × 46 tc; und (6) 1024 × 768 × 24 tc. Diese Prozedur gibt im Erfolgsfall TRUE und andernfalls FALSE aus. Mögliche zurückgegebene Fehler sind: (1) E_MEMTEST: Speichertest der Hardwareeinheit **250** hat versagt; (2) E_ENGINETEST: Ausführungstest der Hardwareeinheit **250** hat versagt; (3) E_PCI: Entweder ist Hardwareeinheit **250** oder Bus **100** nicht vorhanden; (4) E_REGTEST: Hardwareeinheit **250** hat bei dem Test der internen Register versagt; (5) E_NOTSUPPORTED: Modus wird von derzeitiger Hardware nicht unterstützt (z.B. 16 bpp AN für eine bestimmte Hardwareeinheit **250**); und (6) E_PARAMS: Ungültige Parameter für die Prozedur.

[0104] Eine weitere Prozedur in der HDGL **240** ist die RestoreTextMode-Prozedur:
RestoreTextMode()

[0105] Diese Prozedur setzt den Videomodus wieder auf den VGA-Textmodus (in einer Ausführungsform z.B. auf die Modusnummer **3**) zurück. Diese Prozedur wird als Gegenstück zu der InitGraph-Prozedur benutzt, allerdings bleibt der Status der Hardwareeinheit **250** nach diesem Aufruf undefiniert.

[0106] Eine weitere Prozedur in der HDGL **240** ist die DisplayContext * QueryGraphicsDevice-Prozedur:
DisplayContext * QueryGraphicsDevice ()

[0107] Durch diese Prozedur wird die Adresse der systemweiten Anzeigekontextstruktur erhalten, welche die

gegenwärtigen Informationen über den Status der Hardwareeinheit **250** enthält. Die SetPalette-Prozedur ist eine weitere Prozedur in der HDGL **240**:
BOOL SetPalette (WORD int_palette [,Palette* palette, WORD start, WORD count])

[0108] Diese Prozedur initialisiert die Palettenregister auf eine der folgenden Einstellungen (Wert der init_palette ist): (1) PAL_GREY für eine Graustufenpalette; (2) PAL_RGB für Simulation der Indexfarbe (3-3-2); und (3) PAL_CUSTOM für eine Anwenderpalette – in diesem Fall wird der dritte Parameter eingefügt und ist ein Zeiger zu einer Anwenderpalette. Die Argumente 'start' und 'count' definieren den Anfangsindex und die gesamte Anzahl an von der gegebenen Palette einzustellenden Palettenzellen. Für die gesamte Palette beträgt sie in einer Ausführungsform 0 bzw. 256. Die Prozedur gibt im Erfolgsfall TRUE und anderenfalls FALSE aus, wobei mögliche Fehler E_PARAMS für ungültige Parameter sind.

[0109] Die SetQualityDial-Prozedur ist eine weitere Prozedur in der HDGL **240**:
SetQualityDial (BYTE bQuality)

[0110] Diese Prozedur setzt in einer Ausführungsform einen Wert des Qualitäts-/Leistungs-Steuerfeldes auf einen Wert in dem Bereich [0,225], um die Darstellungsleistung und Bildqualität anzupassen. Je niedriger dieser Wert ist, umso schneller sind die Parametrisierungen, jedoch sind Letztere auch weniger präzise. Wenn das derzeitige Bild ein Animationsschritt ist, ist es vorteilhaft, diesen Wert auf eine beliebige kleine Zahl (z.B. weniger als 100) einzustellen, um eine höhere Geschwindigkeit zu bewerkstelligen. Wenn Genauigkeit notwendig ist, wird der Wert auf einen gewissen hohen Wert gestellt (z.B. über 200).

[0111] Parametrisierungsroutinen werden typischerweise mit der Geschwindigkeit als dem signifikantesten Faktor implementiert. Allerdings wird durch diesen Ansatz die Darstellungsqualität beeinträchtigt. Zu Bewerkstelligung eines ausgeglichenen Gleichgewichts ist die Einstell 'Qualität' hinzugefügt worden. Wenn sie auf Null gesetzt ist, verwenden die Darstellungsroutinen den Ansatz mit der höchsten Leistungsrate, der nicht immer der Ansatz mit der höchsten grafischen Präzision ist. Durch eine Erhöhung dieses Werts bis zu dem Maximum wird die Genauigkeit auf die Kosten eines gewissen Verlusts der Leistungsrate erhöht werden. Die Genauigkeit wird durch die Fehlerferme in Z, die Farbberechnungen sowie die Überlappungsflächen der Darstellungsgrundelemente in Abhängigkeit von der Hardwareeinheit **250** widergespiegelt. Mit einem Texture-Mapping wird die folgende Heuristik benutzt: wenn die Textur perspektivisch korrigiert und die vertikale Größe bzw. die Differenz in Z 'klein' ist, wird die Textur linear anstatt perspektivisch gezeichnet werden, um den Parametrisierungsschritt zu beschleunigen.

[0112] Diejenige Prozedur, die die hardwareunabhängigen Stapelverarbeitungszellen des Stapelverarbeitungsfeldes **420** ausliest und die entsprechenden hardwareabhängigen Mikrobefehle innerhalb einer Anzeigeliste **430** im Speicher aufbaut, ist die BuildDisplayList-Prozedur:
BOOL BuildDisplayList (Stapel *p)

[0113] Diese Prozedur setzt einen Zeiger (Stapel *p) auf ein Stapelverarbeitungsfeld **420** vom Stapeltyp, und baut in dem Systemspeicher **102** oder in dem RAM der Hardwareeinheit **250** eine Anzeige auf. Durch eine Reihe dieser Befehle wird eine Anzeigeliste aufgebaut, bis FlushDisplayList für eine Darstellung der Anzeigeliste auf dem Bildschirm **105** aufgerufen wird. Diese Prozedur gibt im Erfolgsfall TRUE und andernfalls FALSE aus.

Exemplarische hardwareabhängige Grafikbibliothek-Texture-Mapping-Prozeduren **330**

[0114] Die Texture-Mapping-Prozeduren der HDGL **240** sind nachstehend in einer exemplarischen Ausführungsform illustriert. Die Prozedur
BOOL TextureHeapAlloc(WORD* pwHeapID, WORD wHeapSize)
weist Speicher von einem Texture-Heap zu. Für einen Host für mehrere Anwender ist es beabsichtigt, verschiedene Verfahren zu unterscheiden und ihre jeweiligen eigenen Texture-Heap-Räume zu verwalten. Das 'wHeapSize'-Feld repräsentiert die Anwenderanfrage für die Heap-Größe. Diese Größe muss gleich oder kleiner als der Wert 'wTextureAvail' sein, der in der DisplayContext-Struktur erhalten wird. Diese Prozeduren geben im Erfolgsfall TRUE und andernfalls FALSE aus. Bei einem Erfolg enthält der Zeiger *pwHeapID das Heap-Handle, das dem neuen Heap-Raum zugewiesen worden ist.

[0115] Die HDGL 240-Prozedur:
BOOL RegisterTexture(WORD *pwTexID, LL_Texture* pTex)
registriert eine Textur. Sämtliche Texture-Informationen werden aus einer Texture-Daten-Struktur abgelesen, die zuvor von dem Anwender gebildet worden ist, nach dem Aufruf jedoch nicht mehr benötigt wird. Die Textu-

re-Daten werden in ein hardwareabhängiges Format umgewandelt. Die neuen Daten werden entweder in dem gleichen Speicher, indem sich die Texture-Quelldaten befinden, zurück gespeichert (z.B. werden die Texture-Quelldaten überschrieben), oder falls ein TEX_PROTECT-Bit in dem 'dwFlag' gesetzt ist, wird ihnen ein neuer Speicherblock zugewiesen, wodurch die ursprünglichen Texture-Quelldaten erhalten werden. Der Zeiger 'pTex' zu der Anwender-Quellen-Texture wird erhalten. Die Identifikationsnummer der registrierten Textur wird in einem Wort gespeichert, auf das durch pwTexID gezeigt wird. Diese Prozedur gibt bei Erfolg TRUE und andernfalls FALSE aus. Im Erfolgsfall enthält der Zeiger *pwTexID das Texture-Handle und N_LIB_MEM wird in 'dwFlags' gesetzt, wenn ein hardwareabhängiges Texture-Format in einem Privatbibliothek-Heap anstatt über die Quellen-Texture-Daten gespeichert wird.

[0116] Die HDGL 240-Prozedur:
BOOL FreeTexture (WORD wTexID)
gibt die Texture-Nummer wTexID frei, die von ihrer Registrierung ausgegeben wurde. Diese Textur kann nicht länger verwendet werden und wird von dem Texture-Bibliothek-Heap freigegeben, wenn IN_LIB_MEM von der Registrierungsprozedur in 'dwFlags' gesetzt worden ist.

[0117] Die HDGL 240-Prozedur:
BOOL LoadTexture (WORD WTEXid)
lädt eine Textur, dessen Handle wTexID ist, in den Texture-Speicher der Hardwareeinheit **250**. Typischerweise wird eine Textur geladen, bevor ihre wTexID in dem wTexID-Teil einer Stapelverarbeitungsstruktur verwendet wird. In einer Ausführungsform kann diese Prozedur aufgrund der Speicherfragmentierung, die durch mehrere Lade- und Entladevorgänge von Texturen verursacht worden ist, einen Fehler ausgeben. In diesem Fall wird die HeapCollect-Prozedur für eine Speicherbereinigung mit einem Neuversuch von LoadTexture verwendet. Allerdings kann das Makro AutoHeapCollect(True) dazu benutzt werden, die Bibliothek in einen Modus zu versetzen, in dem eine Speicherbereinigung im Falle eines nicht erfolgreichen Ladevorgangs automatisch ausgeführt wird. Diese Prozedur gibt im Erfolgsfall TRUE und im anderen Fall FALSE aus.

[0118] Die HDGL 240-Prozedur:
BOOL UnloadTexture (WORD wTexID)
entlädt eine Textur, dessen Handle wTexID von dem privaten Speicher der Hardwareeinheit **250** ist. Diese Prozedur gibt die Textur nicht "frei"; d.h. die Textur ist immer noch registriert und kann später erneut geladen werden. Die Prozedur gibt im Erfolgsfall TRUE und andernfalls FALSE aus. Die HDGL 240-Prozedur:
HeapCollect (WORD wHeapID)
führt eine Speicherbereinigung der Texture-Daten in dem privaten Speicher der Hardwareeinheit **250** aus. Diese Prozedur wird verwendet, wenn LoadTexture aufgrund einer Speicherfragmentierung versagt. Das Argument 'wHeapID' ist das Heap-Handle des defragmentierten Heaps. Dieses Handle wird durch ein Anruf der TextureHeapAllocQ erhalten.

[0119] Ein Heap-Bereinigungsmakro wird durch diese Implementierung der HDGL **240** ebenfalls verwendet:
AutoHeapCollect (WORD wHeapID, BOOL collect)

[0120] Diese Prozedur stellt die automatische Heap-Sammlung auf AN oder AUS, um Texturen in den privaten Texture-Speicher der Hardwareeinheit **250** zu laden. Das Argument 'wHeapID' ist das Heap-Handle der betroffenen Heaps. Dieses Handle wird durch ein Aufruf von TextureHeapAlloc() erhalten.

[0121] In der bevorzugten Ausführungsform der vorliegenden Erfindung stellt eine hardwareabhängige Low-Level-Grafikbibliothek eine Schnittstelle zwischen einer hardwareunabhängigen High-Level-Grafikbibliothek und einer Grafikhardwareeinheit her, die wie oben erläutert flexible und effiziente Grafikdarstellungsprozeduren aufweist. Obgleich die vorliegende Erfindung durch bestimmte Ausführungsformen beschrieben worden ist, sollte sich verstehen, dass die vorliegende Erfindung nicht durch derartige Ausführungsformen, sondern lediglich durch die beiliegenden Ansprüche begrenzt sein sollte.

Patentansprüche

1. Computergesteuertes grafisches Anzeigesystem, versehen mit:
einem mit einem Bus gekoppelten Prozessor;
einer Speichereinheit zum Speichern von Informationen;
einer Hardware-Grafikeinheit, die hardwareabhängige Mikrobefehle von einer in der Speichereinheit gespeicherten Anzeigeliste erhält und ein Bild auf einem Bildschirm erzeugt;
einer High-Level-Grafikbibliothek, die hardwareunabhängige Grafikdarstellungsprozeduren, die von dem Pro-

zessor ausgeführt werden, umfasst, wobei die hardwareunabhängigen Grafikdarstellungsprozeduren zum Verarbeiten von Grafikdarstellungsanfragen von einer High-Level-Anwendung dienen, um hardwareunabhängige Ausgangsdatenstrukturen, die Grafikoperanden umfassen, zu erzeugen; und
einer hardwareabhängigen Low-Level-Grafikbibliothek, die von dem Prozessor ausgeführt wird, um die hardwareunabhängigen Ausgangsdatenstrukturen zu verarbeiten, um aus diesen die Mikrobefehle für die Hardware-Grafikeinheit zu erzeugen, wobei die High-Level-Grafikbibliothek ohne Umgestaltung mit einer Vielzahl von unterschiedlichen Hardware-Grafikeinheiten kompatibel ist.

2. System gemäß Anspruch 1, bei welchem die hardwareunabhängigen Ausgangsdatenstrukturen ein Feld von Stapelverarbeitungszellen umfassen, wobei jede Stapelverarbeitungszelle eine separate auszuführende Grafikoperation darstellt und wobei das Feld von Stapelverarbeitungszellen nacheinander an die zu verarbeitende hardwareabhängige Low-Level-Grafikbibliothek geleitet wird, um die Mikrobefehle zu erzeugen.

3. System nach Anspruch 2, bei welchem die Speichereinheit einen Codecache aufweist und bei welchem die hardwareabhängige Low-Level-Grafikbibliothek Parametrisierungsprozeduren umfasst, die von dem Prozessor ausgeführt werden, um die Mikrobefehle zu erzeugen, wobei das Feld von Stapelverarbeitungszellen ohne Unterbrechung durch die Parametrisierungsprozeduren verarbeitet werden, um Cachefehler zu vermeiden.

4. System nach Anspruch 2, bei welchem die hardwareabhängige Low-Level-Grafikbibliothek Parametrisierungsprozeduren zum Verarbeiten von Polygon-Grundelementen, Sätzen von Grafiklinien und Sätzen von Grafikpunkten aufweist.

5. System nach Anspruch 4, bei welchem die Parametrisierungsprozeduren ferner dem Verarbeiten von Bitleveltransfers, Füllungen und Umsetzungen zwischen Texture-Map-Formaten dient.

6. System nach Anspruch 2, bei welchem die hardwareabhängige Low-Level-Grafikbibliothek ferner eine Leistungs-/Qualitäts-Einstellprozedur aufweist, die von dem Prozessor ausgeführt wird, um die Darstellungsleistungsrate einzustellen und entsprechend die Darstellungsqualität des auf dem Bildschirm dargestellten Bildes einzustellen.

7. System nach Anspruch 6, bei welchem die Leistungs-/Qualitäts-Einstellungsprozedur, die von dem Prozessor ausgeführt wird, dem Einstellen des Pegels von linearen und perspektivischen Unterteilungen, die zur Grafikdarstellung ausgeführt werden, sowie zum Einstellen des Pegels von für Polygonüberlappungen benutzte Fehlerkorrekturfaktoren sowie zum Einstellen der Grundelementgrenzgröße zwecks Abschaltung der Perspektivendarstellung dient.

8. Computergesteuertes grafisches Anzeigesystem, versehen mit:
einem mit einem Adressen-/Daten-Bus gekoppelten Prozessor;
einer Speichereinheit zum Speichern von Grafikinformationen;
einer Grafikhardwareeinheit zum Verarbeiten von in einer Anzeigeliste gespeicherten hardwareabhängigen Mikrobefehlen und, in Ansprechen hierauf, zum Erzeugen eines Bildes auf einem Bildschirm;
einer High-Level-Grafikbibliothek mit von dem Prozessor ausgeführten hardwareunabhängigen Grafikdarstellungsprozeduren, wobei die hardwareunabhängigen Grafikdarstellungsprozeduren dem Erhalten von Grafikdarstellungsanfragen von einer High-Level-Anwendung und zum Erzeugen eines hardwareunabhängigen Feldes von Stapelverarbeitungszellen von diesen dient, wobei jede Stapelverarbeitungszelle eine individuelle Grafikdarstellungsoperation darstellt, die ein Darstellungsgrundelement einschließt; und
einer von dem Prozessor ausgeführten hardwareabhängigen Low-Level-Grafikbibliothek zur Aufnahme des Feldes von Stapelverarbeitungszellen von den Grafikdarstellungsprozeduren, wobei die hardwareabhängige Low-Level-Grafikbibliothek dem Parametrisieren von Zellen des Feldes von Stapelverarbeitungszellen dient, um die Mikrobefehle für die Hardware-Grafikeinheit zu erzeugen.

9. System nach Anspruch 8, bei welchem die Speichereinheit einen Codecache und einen Datencache aufweist und wobei die hardwareabhängige Low-Level-Grafikbibliothek Parametrisierungsprozeduren umfasst, die in dem Codecache gespeichert sind und an Zellen des Feldes von Stapelverarbeitungszellen, die in dem Datencache gespeichert sind, ausgeführt werden, um die Mikrobefehle zu erzeugen, wobei die Zellen des Feldes von Stapelverarbeitungszellen ohne Unterbrechung durch die Parametrisierungsprozeduren ausgeführt werden, um Codecachefehlgriffe zu vermeiden.

10. System gemäß Anspruch 8, bei welchem die hardwareabhängige Low-Level-Grafikbibliothek von dem

Prozessor ausgeführte Parametrisierungsprozeduren umfasst, um Polygon-Grundelemente, Sätze von Grafiklinien und Sätze von Grafikpunkten zu verarbeiten.

11. System gemäß Anspruch 10, bei welchem die Parametrisierungsprozeduren ferner dem Verarbeiten von Bitleveltransfers, Füllungen und dem Durchführen von Texture-Map-Format-Umsetzungen dienen.

12. System gemäß Anspruch 8, bei welchem die hardwareabhängige Low-Level-Grafikbibliothek ferner eine Leistungs-/Qualitäts-Einstellprozedur aufweist, die von dem Prozessor ausgeführt wird, um die Darstellungsleistungsrate einzustellen und entsprechend die Darstellungsqualität des auf dem Bildschirm dargestellten Grafikbildes einzustellen.

13. System nach Anspruch 12, bei welchem die Leistungs-/Qualitäts-Einstellungsprozedur, die von dem Prozessor ausgeführt wird, dem Einstellen des Pegels von linearen und perspektivischen Unterteilungen, die zur Grafikdarstellung ausgeführt werden, sowie zum Einstellen des Pegels von für Polygonüberlappungen benutzte Fehlerkorrekturfaktoren, sowie zum Einstellen der Grundelementgrenzgröße zur Abschaltung der Perspektivendarstellung dient.

14. Verfahren für ein computergesteuertes Grafiksystem mit einem mit einem Bus gekoppelten Prozessor, einer Speichereinheit zum Speichern von Informationen und einer Grafikhardwareeinheit zum Darstellen von Bildern auf einem Bildschirm basierend auf Mikrobefehlen innerhalb einer Anzeigeliste, wobei das Verfahren dem Aufbau der Anzeigeliste dient, welches die nachstehenden computerimplementierten Schritte umfasst:
Erzeugen eines Satzes von Grafikdarstellungsanforderungen einschließlich Anforderungen zum Darstellen von Grafikgrundelementen einschließlich Polygonen, Linien und Punkten unter Verwendung einer von dem Prozessor ausgeführten High-Level-Anwendung;
Umsetzung des Satzes von Grafikdarstellungsanfragen in ein hardwareunabhängiges Feld von Stapelverarbeitungszellen unter Verwendung von hardwareabhängigen Prozeduren einer High-Level-Grafikbibliothek, die von dem Prozessor ausgeführt wird, wobei jede Stapelverarbeitungszelle eine individuelle Grafikoperation umfasst;
Aufnahme des hardwareunabhängigen Feldes von Stapelverarbeitungszellen und Erzeugen daraus einer hardwareabhängigen Anzeigelisten von Mikrobefehlen durch sequentielles Verarbeiten von Zellen des Feldes von Stapelverarbeitungszellen unter Verwendung einer hardwareabhängigen Low-Level-Grafikbibliothek, die von dem Prozessor ausgeführt wird; und
Zugreifen auf die Anzeigeliste und Anzeigen eines Bildes auf dem Bildschirm unter Verwendung der Grafikhardwareeinheit, wobei die High-Level-Grafikbibliothek ohne Umgestaltung mit einer Vielzahl von unterschiedlichen Grafikhardwareeinheiten kompatibel ist.

15. Verfahren gemäß Anspruch 14, bei welchem im Zuge des Schrittes des Aufnehmens des hardwareunabhängigen Feldes von Stapelverarbeitungszellen und des Erzeugens daraus einer hardwareabhängigen Anzeigeliste von Mikrobefehlen:
innerhalb der Stapelverarbeitungszellen Polygon-Grundelemente verarbeitet werden, um daraus Anzeigelisten-Mikrobefehle zu erzeugen;
Sätze von Linien innerhalb der Stapelverarbeitungszellen verarbeitet werden, um daraus Anzeigenlistenmikrobefehle zu erzeugen; und
Sätze von Punkten innerhalb der Stapelverarbeitungszellen verarbeitet werden, um daraus Anzeigenlistenmikrobefehle zu erzeugen.

16. Verfahren gemäß Anspruch 15, bei welchem im Zuge des Schrittes des Aufnehmens des hardwareunabhängigen Feldes von Stapelverarbeitungszellen und des Erzeugens daraus einer hardwareabhängigen Anzeigeliste von Mikrobefehlen ferner:
Bitleveltransfers verarbeitet werden, um daraus Anzeigelistenmikrobefehle zu erzeugen;
Fülloperationen verarbeitet werden, um daraus Anzeigelistenmikrobefehle zu erzeugen und
Texture-Map-Umsetzungen verarbeitet werden, um ein Texture-Map von einem Anzeigeformat in ein anders umzusetzen.

17. Verfahren gemäß Anspruch 14, bei welchem im Zuge des Schrittes des Aufnehmens des hardwareunabhängigen Feldes von Stapelverarbeitungszellen und des Erzeugens daraus einer hardwareabhängigen Anzeigeliste von Mikrobefehlen:
Parametrisierungsprozeduren der hardwareabhängigen Grafikbibliothek in eine Cachespeichereinheit geladen werden; und
Cachefehlgriffe während der Verarbeitung des Feldes von Stapelverarbeitungszellen verhindert werden, indem

die Parametrisierungsprozeduren sequentiell von der Cacheeinheit an den Zellen des Feldes von Stapelverarbeitungszellen ohne Unterbrechung ausgeführt werden.

18. Verfahren gemäß Anspruch 15, bei welchem ferner:
eine Einstellung eines Leistungs-/Qualitäts-Steuerfeldes, welches auf dem Bildschirm angezeigt wird, eingestellt wird;
die Leistungsrate der Grafikhardwareeinheit basierend auf dieser Einstellung erhöht und erniedrigt wird; und
entsprechend die Anzeigequalität der Grafikhardwareeinheit basierend auf der Einstellung erhöht und erniedrigt wird.

19. Verfahren gemäß Anspruch 18, bei welchem im Zuge der Schritte des Erhöhens und Erniedrigens der Anzeigeleistung der Hardwaregrafikeinheit und des entsprechenden Erhöhens und Erniedrigens der Anzeigequalität der Grafikhardwareeinheit basierend auf der besagten Einstellung ferner:
der Pegel von linearen und perspektivischen Unterteilungen, die zur Grafikdarstellung ausgeführt werden, eingestellt wird;
der Pegel von Fehlerkorrekturfaktoren eingestellt wird, die für Polygonüberlappungen verwendet werden; und
die Grundelementgrenzgröße zur Abschaltung der Perspektivendarstellung eingestellt wird.

20. Verfahren gemäß Anspruch 15, bei welchem die Grafikdarstellungsanfragen ferner Anzeigeoperationen von Texture-Maps umfassen.

Es folgen 13 Blatt Zeichnungen

Anhängende Zeichnungen

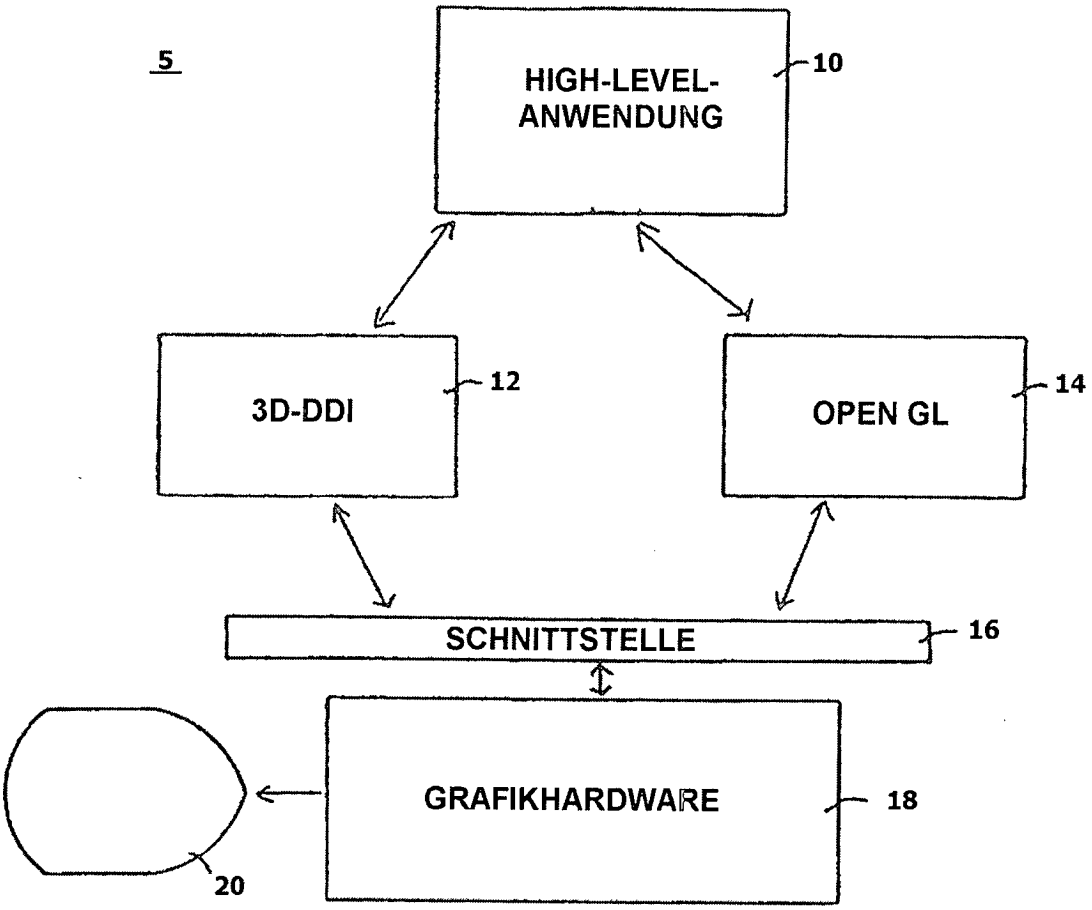

FIG. 1A
(Stand der Technik)

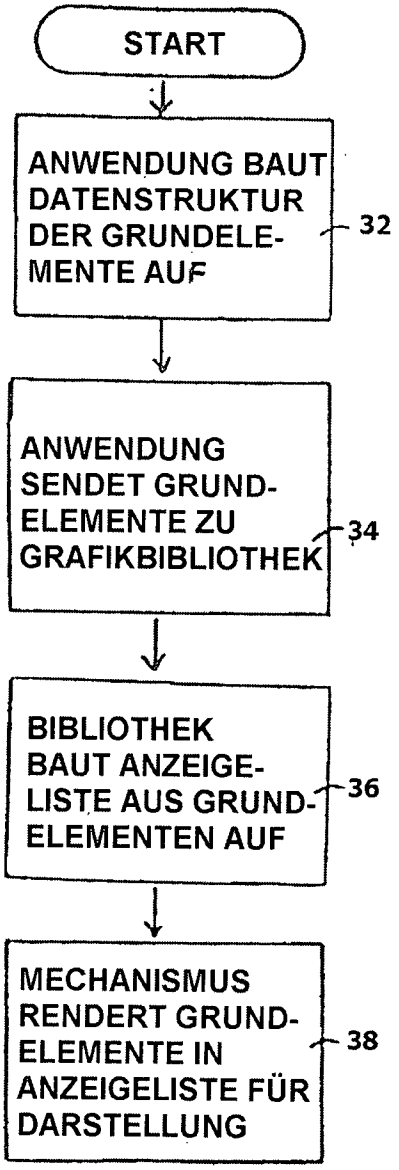

FIG. 1B
(Stand der Technik)

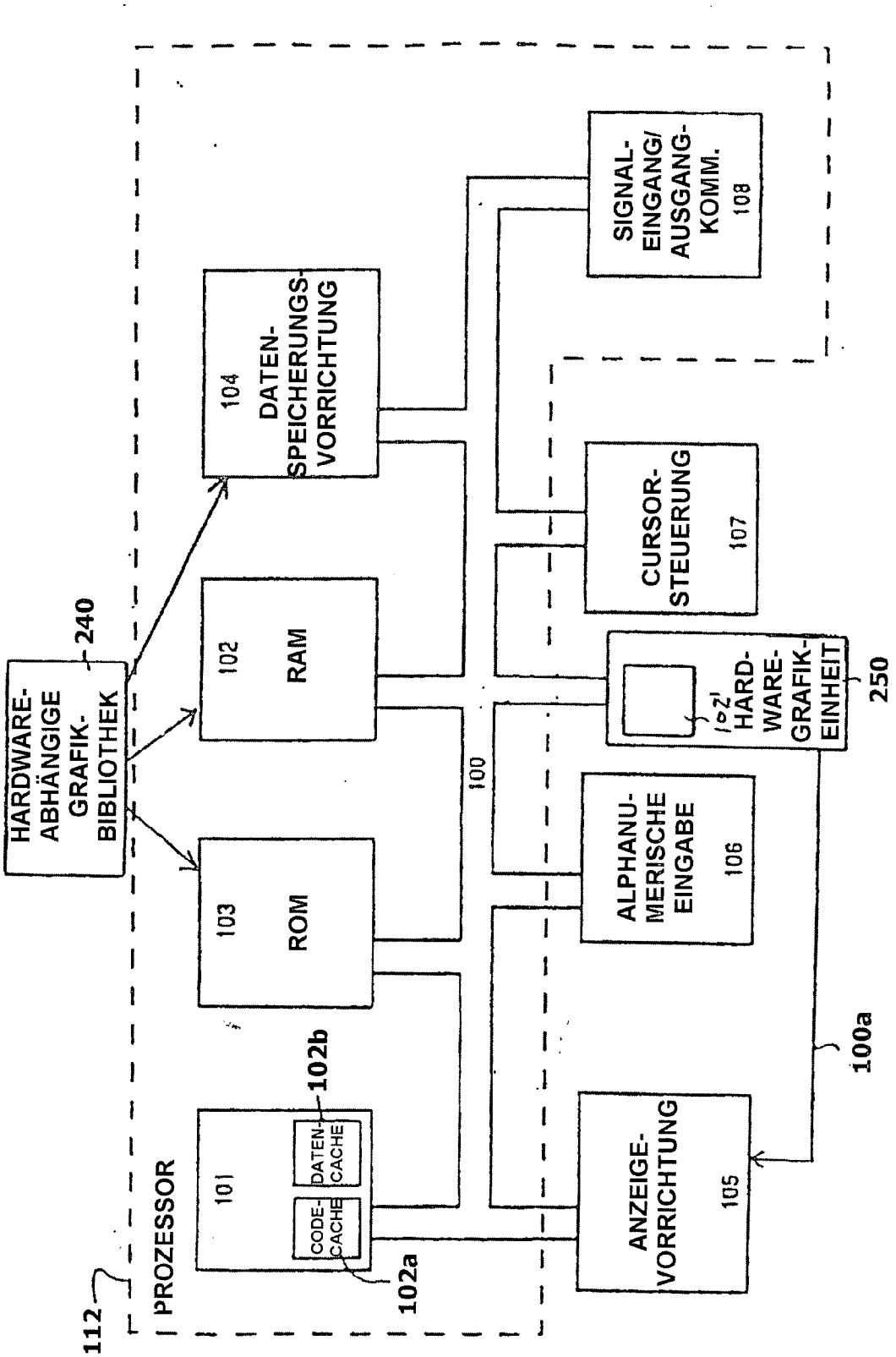
HARDWARE-ABHÄNGIGE GRAFIK-BIBLIOTHEK
240
104 DATEN-SPEICHERUNGS-VORRICHTUNG
102 RAM
103 ROM
101 PROZESSOR
112
102b
DATEN-CACHE
CODE-CACHE
102a
100
SIGNAL-EINGANG/ AUSGANG-KOMM. 108
CURSOR-STEUERUNG 107
102' HARD-WARE-GRAFIK-EINHEIT
250
ALPHANU-MERISCHE EINGABE 106
ANZEIGE-VORRICHTUNG 105
100a

FIG. 2

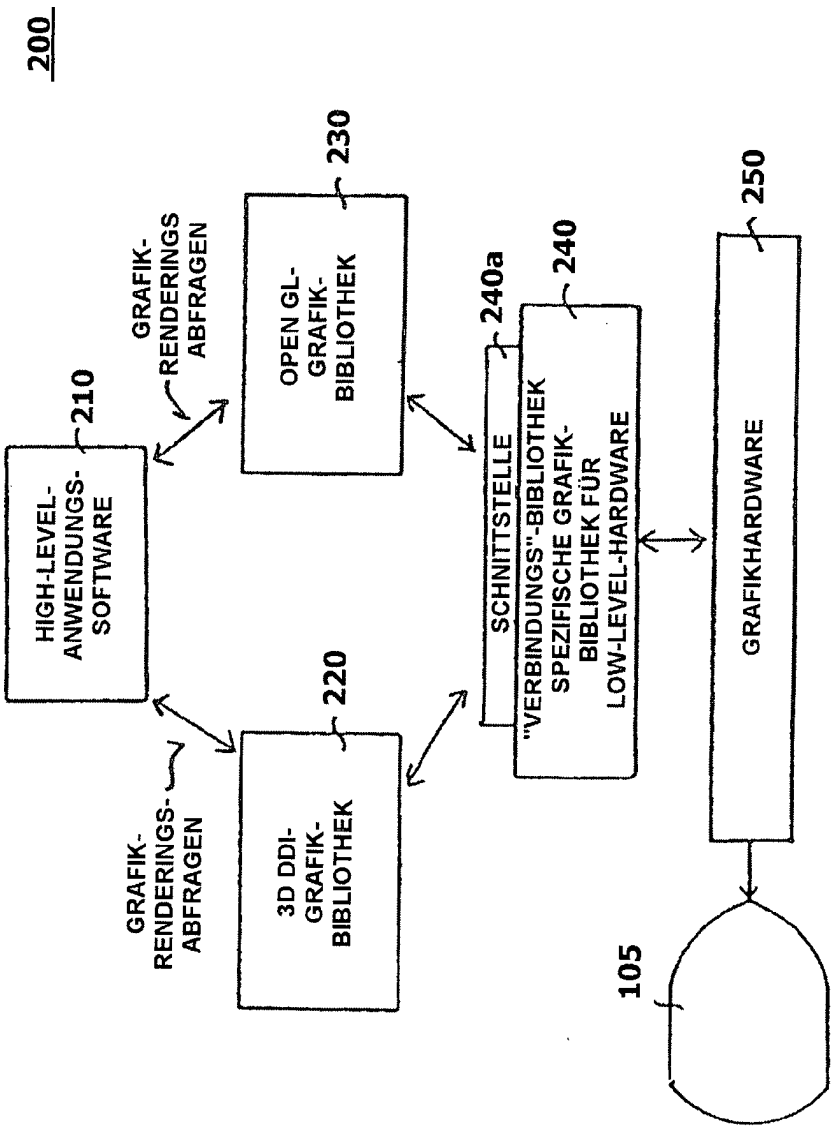
200
HIGH-LEVEL-ANWENDUNGS-SOFTWARE
210
GRAFIK-RENDERINGS ABFRAGEN
OPEN GL-GRAFIK-BIBLIOTHEK
230
GRAFIK-RENDERINGS-ABFRAGEN
3D DDI-GRAFIK-BIBLIOTHEK
220
SCHNITTSTELLE
240a
"VERBINDUNGS"-BIBLIOTHEK SPEZIFISCHE GRAFIK-BIBLIOTHEK FÜR LOW-LEVEL-HARDWARE
240
GRAFIKHARDWARE
250
105

FIG. 3

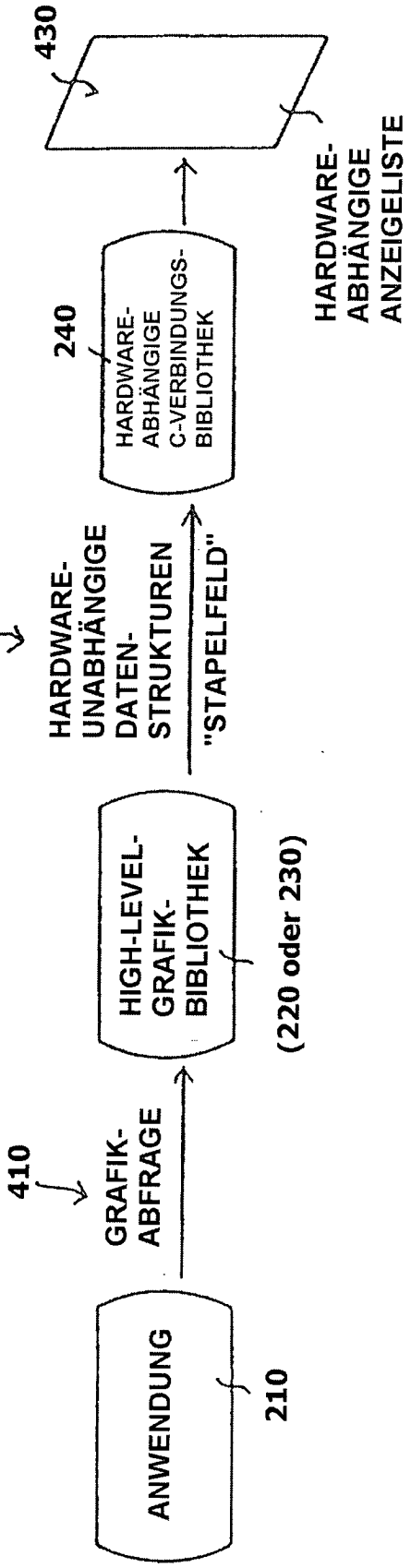
ANWENDUNG
210
410
GRAFIK-ABFRAGE
HIGH-LEVEL-GRAFIK-BIBLIOTHEK
(220 oder 230)
420
HARDWARE-UNABHÄNGIGE DATEN-STRUKTUREN
"STAPELFELD"
240
HARDWARE-ABHÄNGIGE C-VERBINDUNGS-BIBLIOTHEK
430
HARDWARE-ABHÄNGIGE ANZEIGELISTE

FIG. 4

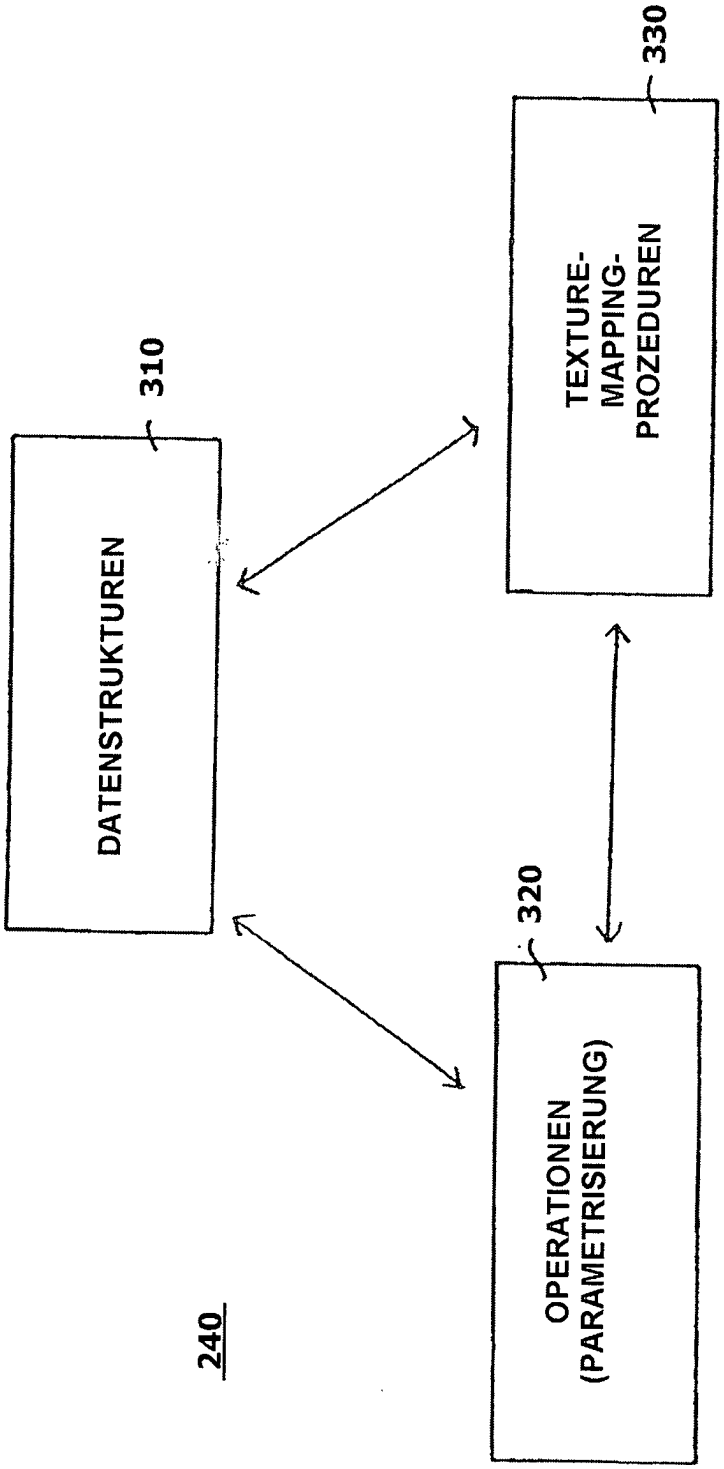
330
TEXTURE-MAPPING-PROZEDUREN
310
DATENSTRUKTUREN
320
OPERATIONEN (PARAMETRISIERUNG)
240

FIG. 5

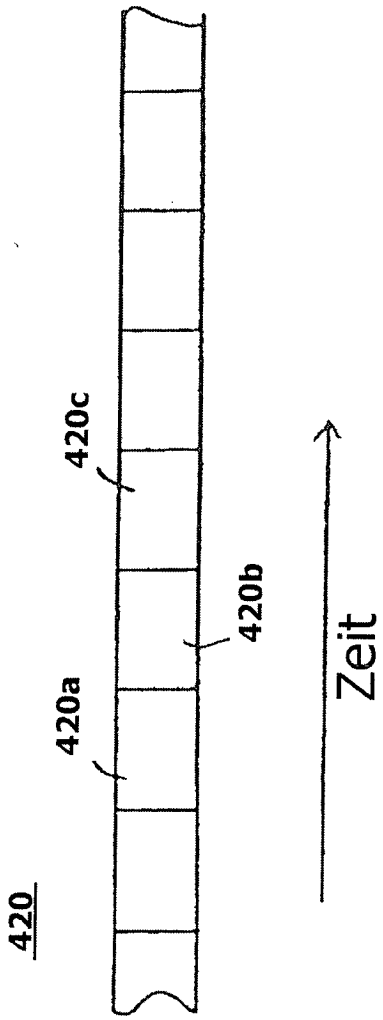

FIG. 6

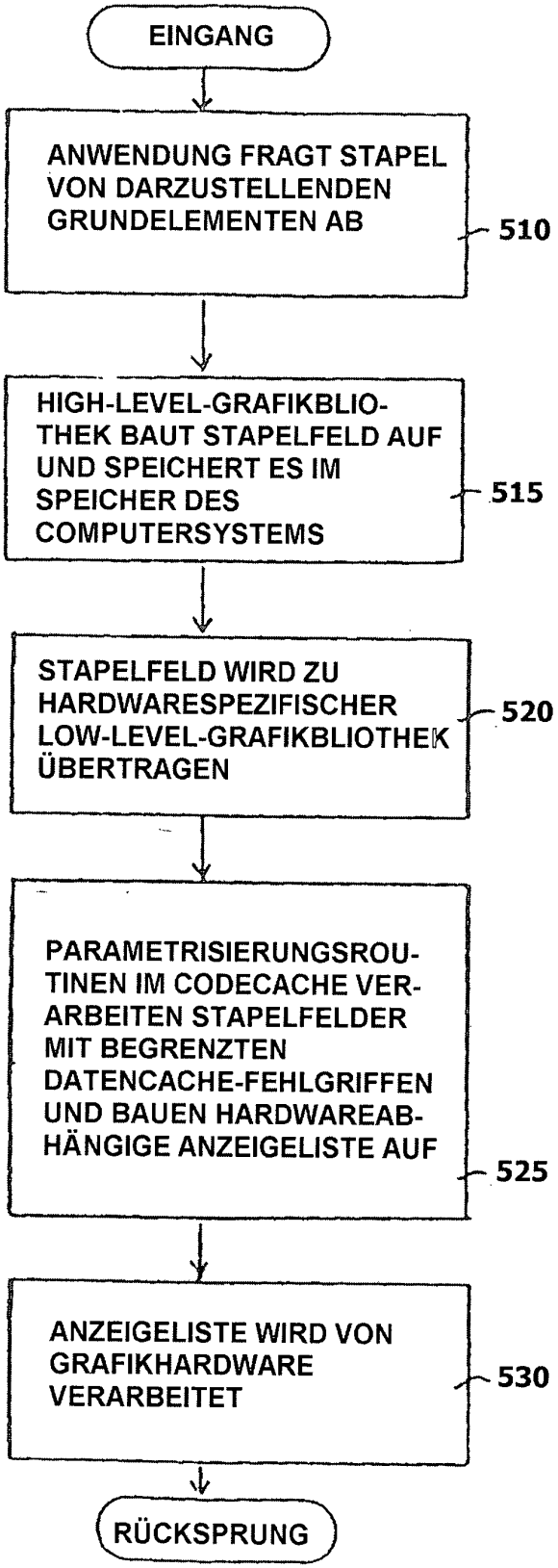
500
EINGANG
ANWENDUNG FRAGT STAPEL VON DARZUSTELLENDEN GRUNDELEMENTEN AB
510
HIGH-LEVEL-GRAFIKBLIOTHEK BAUT STAPELFELD AUF UND SPEICHERT ES IM SPEICHER DES COMPUTERSYSTEMS
515
STAPELFELD WIRD ZU HARDWARESPEZIFISCHER LOW-LEVEL-GRAFIKBLIOTHEK ÜBERTRAGEN
520
PARAMETRISIERUNGSROUTINEN IM CODECACHE VERARBEITEN STAPELFELDER MIT BEGRENZTEN DATENCACHE-FEHLGRIFFEN UND BAUEN HARDWAREABHÄNGIGE ANZEIGELISTE AUF
525
ANZEIGELISTE WIRD VON GRAFIKHARDWARE VERARBEITET
530
RÜCKSPRUNG

FIG. 7

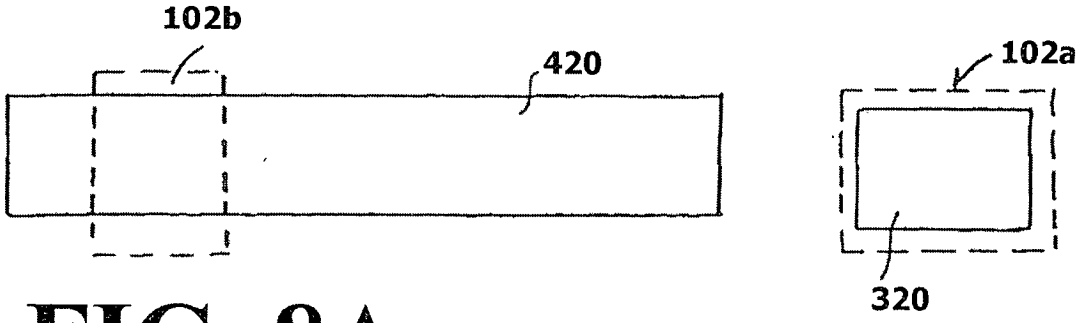

FIG. 8A

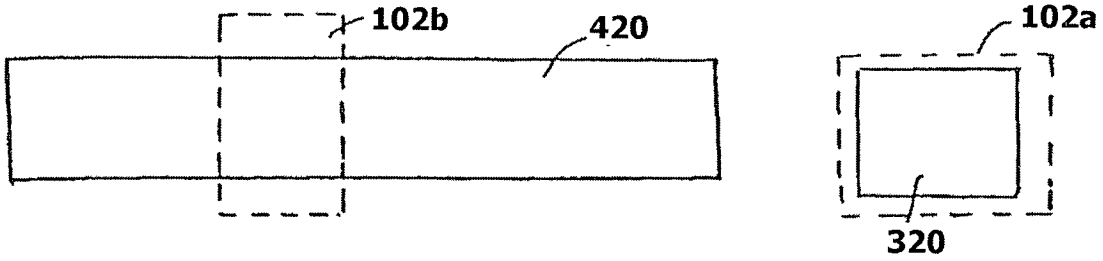

FIG. 8B

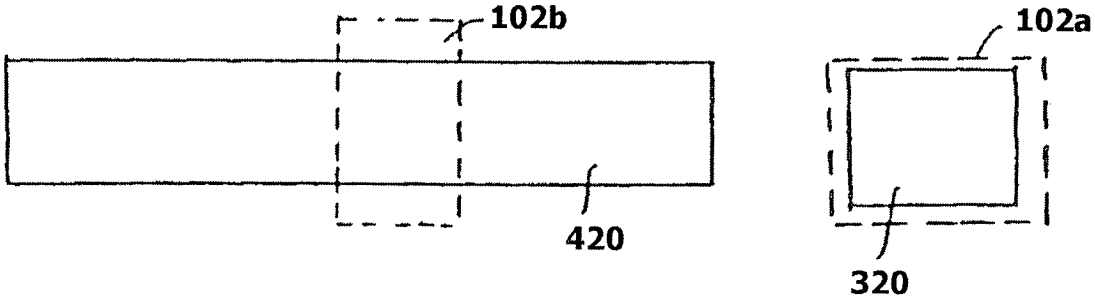

FIG. 8C

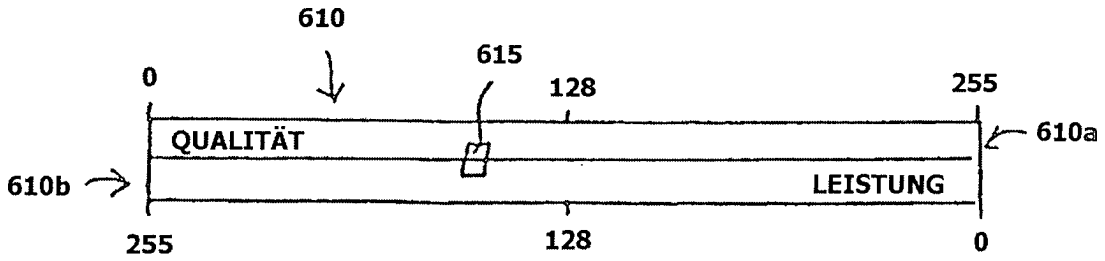
610
615
0
128
255
QUALITÄT
610a
610b
LEISTUNG
255
128
0

FIG. 9A

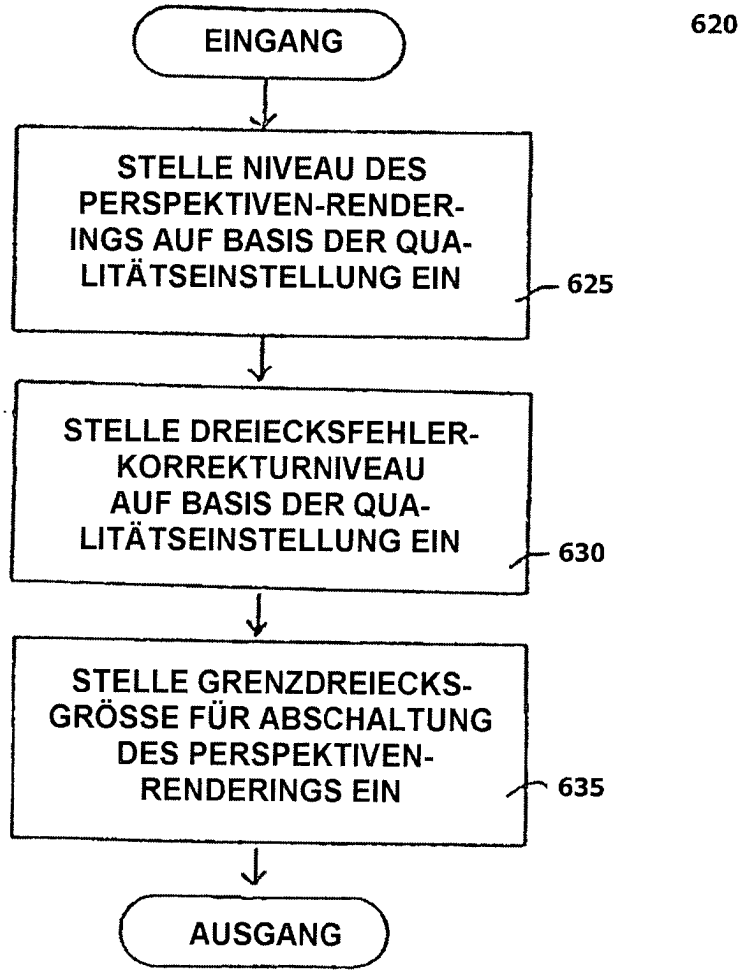
620
EINGANG
STELLE NIVEAU DES PERSPEKTIVEN-RENDERINGS AUF BASIS DER QUALITÄTSEINSTELLUNG EIN
625
STELLE DREIECKSFEHLERKORREKTURNIVEAU AUF BASIS DER QUALITÄTSEINSTELLUNG EIN
630
STELLE GRENZDREIECKSGRÖSSE FÜR ABSCHALTUNG DES PERSPEKTIVENRENDERINGS EIN
635
AUSGANG

FIG. 9B

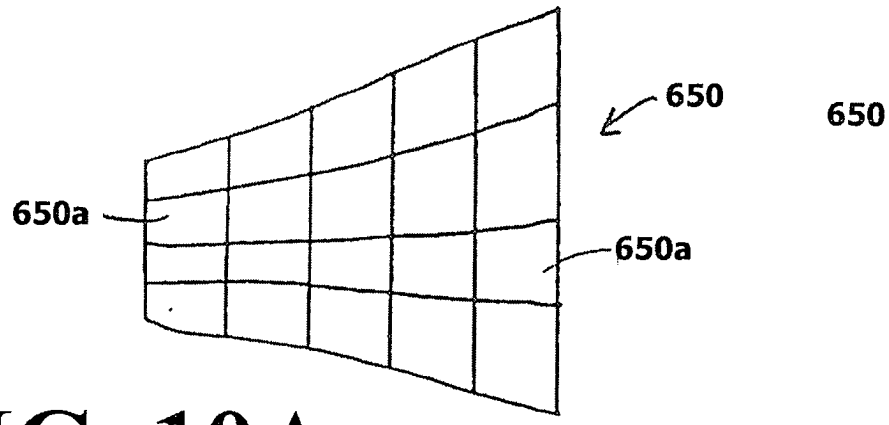

FIG. 10A

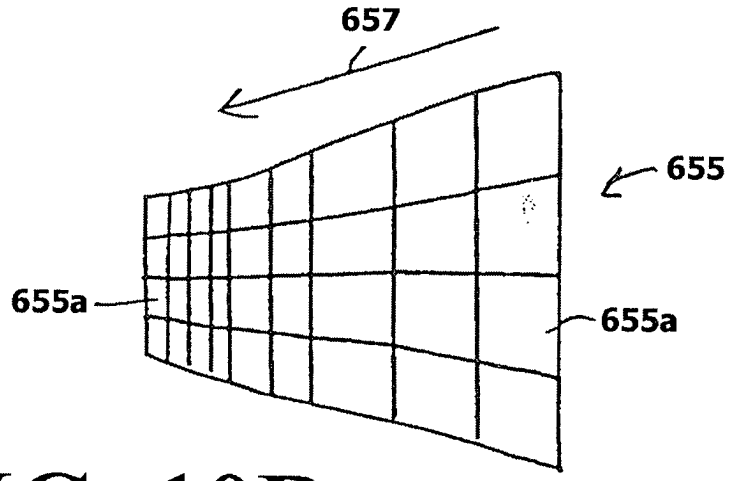

FIG. 10B

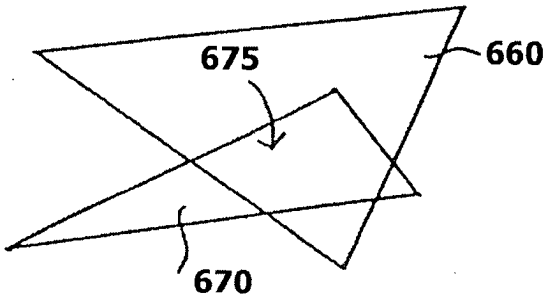

FIG. 10C

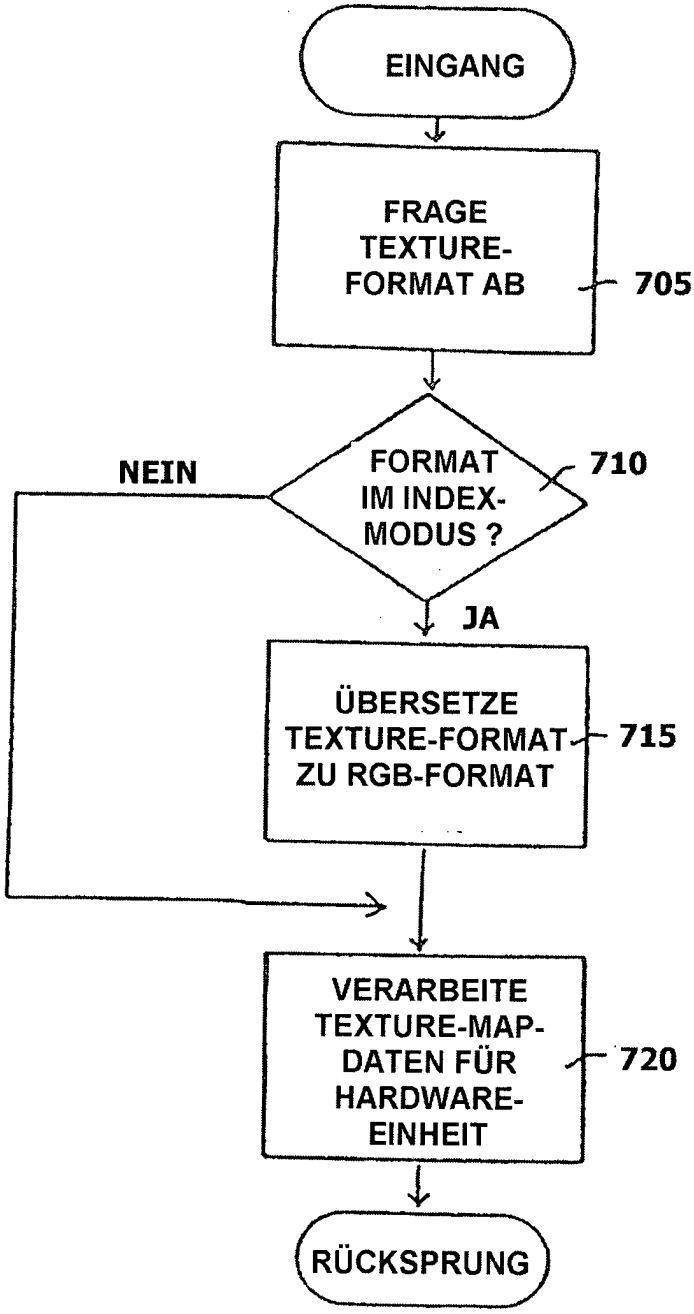
EINGANG
700
FRAGE TEXTURE-FORMAT AB
705
FORMAT IM INDEX-MODUS ?
710
NEIN
JA
ÜBERSETZE TEXTURE-FORMAT ZU RGB-FORMAT
715
VERARBEITE TEXTURE-MAP-DATEN FÜR HARDWARE-EINHEIT
720
RÜCKSPRUNG

FIG. 11

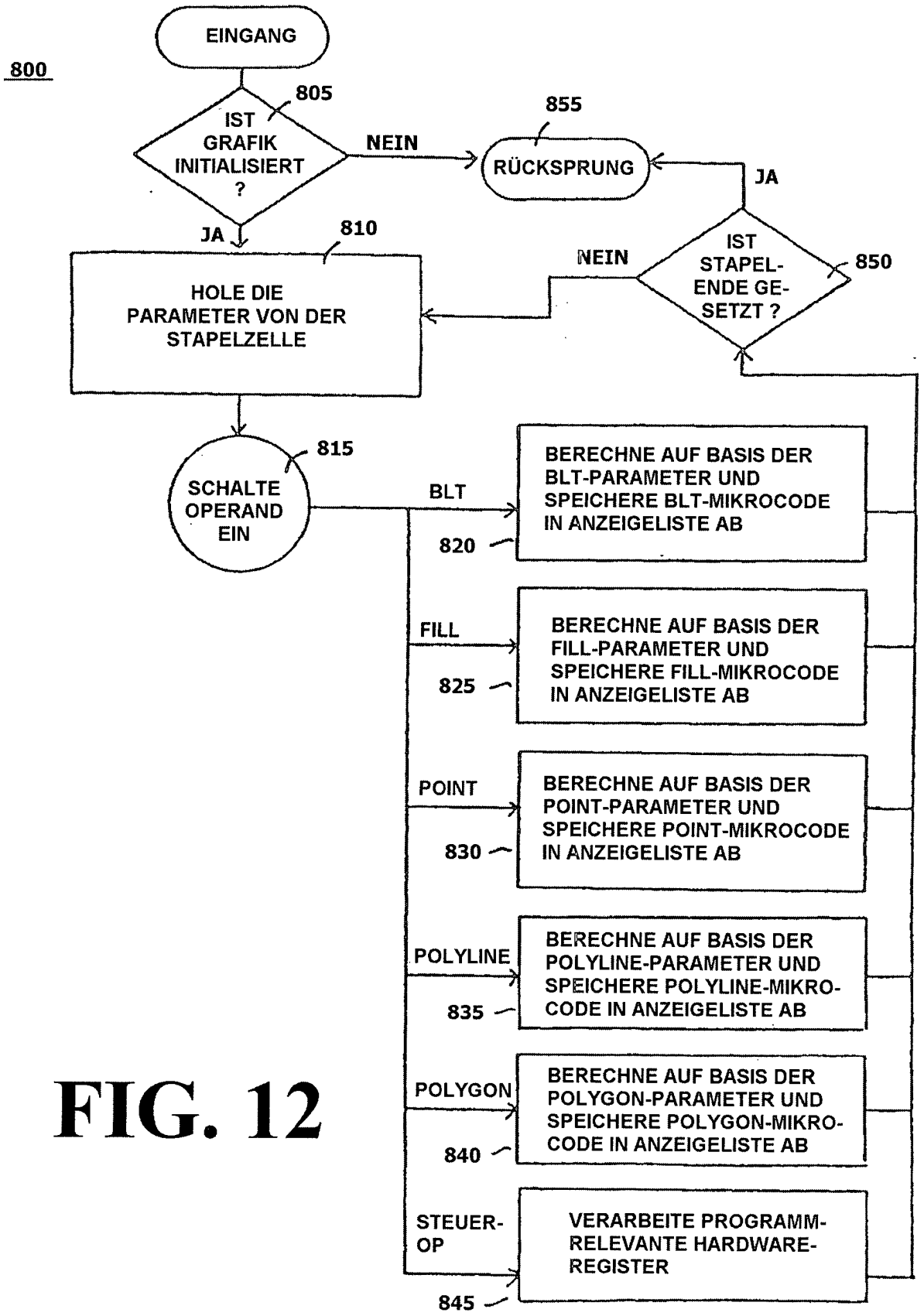
800
EINGANG
805
IST GRAFIK INITIALISIERT ?
NEIN
855
RÜCKSPRUNG
JA
JA
810
HOLE DIE PARAMETER VON DER STAPELZELLE
NEIN
IST STAPEL-ENDE GE-SETZT ?
850
815
SCHALTE OPERAND EIN
BLT
820
BERECHNE AUF BASIS DER BLT-PARAMETER UND SPEICHERE BLT-MIKROCODE IN ANZEIGELISTE AB
FILL
825
BERECHNE AUF BASIS DER FILL-PARAMETER UND SPEICHERE FILL-MIKROCODE IN ANZEIGELISTE AB
POINT
830
BERECHNE AUF BASIS DER POINT-PARAMETER UND SPEICHERE POINT-MIKROCODE IN ANZEIGELISTE AB
POLYLINE
835
BERECHNE AUF BASIS DER POLYLINE-PARAMETER UND SPEICHERE POLYLINE-MIKRO-CODE IN ANZEIGELISTE AB
POLYGON
840
BERECHNE AUF BASIS DER POLYGON-PARAMETER UND SPEICHERE POLYGON-MIKRO-CODE IN ANZEIGELISTE AB
STEUER-OP
845
VERARBEITE PROGRAMM-RELEVANTE HARDWARE-REGISTER

FIG. 12