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

(10) **DE 698 20 579 T2** 2004.10.28

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 923 048 B1**
(21) Deutsches Aktenzeichen: **698 20 579.0**
(96) Europäisches Aktenzeichen: **98 114 114.6**
(96) Europäischer Anmeldetag: **28.07.1998**
(97) Erstveröffentlichung durch das EPA: **16.06.1999**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **17.12.2003**
(47) Veröffentlichungstag im Patentblatt: **28.10.2004**

(51) Int Cl.[7]: **G06T 3/40**

(30) Unionspriorität:
989998 **12.12.1997** **US**

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

(74) Vertreter:
Schoppe, Zimmermann, Stöckeler & Zinkler, 82049 Pullach

(84) Benannte Vertragsstaaten:
DE, GB

(72) Erfinder:
Vondran, Gary L. Jr., Winchester, US

(54) Bezeichnung: **Tetrahaedrische und geschnitten-tetrahaedrische interpolationsvorrichtung**

Anmerkung: Innerhalb von neun Monaten nach der Bekanntmachung des Hinweises auf die Erteilung des europäischen Patents kann jedermann beim Europäischen Patentamt gegen das erteilte europäische Patent Einspruch einlegen. Der Einspruch ist schriftlich einzureichen und zu begründen. Er gilt erst als eingelegt, wenn die Einspruchsgebühr entrichtet worden ist (Art. 99 (1) Europäisches Patentübereinkommen).

Die Übersetzung ist gemäß Artikel II § 3 Abs. 1 IntPatÜG 1991 vom Patentinhaber eingereicht worden. Sie wurde vom Deutschen Patent- und Markenamt inhaltlich nicht geprüft.

Beschreibung

Gebiet der Erfindung

[0001] Diese Erfindung bezieht sich auf die Transformation von Daten und insbesondere auf die Transformation von Daten von einem ersten Raum zu einem zweiten Raum, wie z. B. bei der Umwandlung von einem ersten Farbraum zu einem zweiten Farbraum.

Hintergrund der Erfindung

[0002] Die Kolorimetrie bzw. Farbmessung ist seit langem als eine komplexe Wissenschaft anerkannt. Im allgemeinen hat es sich als möglich und praktisch herausgestellt, Farbstimulivektoren in einem dreidimensionalen Raum darzustellen, der als Tristimulusraum bezeichnet wird. Wie es 1931 durch die CIE (Commission Internationale L'Eclairage) definiert wurde, können im wesentlichen drei Primärfarben (X, Y, Z) kombiniert werden, um alle Lichtwahrnehmungen zu definieren, die wir mit unseren Augen erfahren (d. h. die Farbübereinstimmungseigenschaften eines idealen Dreifarbenbetrachters definiert durch Spezifizieren von drei unabhängigen Funktionen der Wellenlänge, die durch die Farbübereinstimmungsfunktionen des idealen Beobachters identifiziert werden, bilden einen internationalen Standard zum Spezifizieren von Farbe). Die Grundlagen solcher dreidimensionaler Aufbauten sind in der Literatur erörtert, wie z. B. Principles of Color Technology von Billmeyer und Saltzman, veröffentlicht von John Wiley & Sons, Inc., NY, Copyright 1981 (2. Ausg.), und Color Science: Concepts and Methods, Quantitative Data and Formulae, von Wyszecki und Stiles, veröffentlicht von John Wiley & Sons., Inc., Copyright 1982 (2. Ausg.), insbesondere Seiten 119–130.

[0003] Trichromatische Modellsysteme – wie z. B. Rot, Grün, Blau (RGB); Cyan, Magenta, Gelb (CMY), Farbton, Sättigung, Wert (HSV); Farbton, Helligkeit, Sättigung (HLS); Luminanz, Rot-Gelb-Skala, Grün-Blau-Skala (La*b*); Luminanz, Rot-Grün-Skala, Gelb-Blau-Skala (Luv); YIQ, verwendet bei kommerziellem Farbfernsehrundfunk; und dergleichen – liefern Alternativen für den Systementwickler. Siehe z. B. Arbeiten wie Fundamentals of Interactive Computer Graphics, von Foley und Van Dam, Addison-Wesley Publishing Company, insbesondere Seiten 606–621, die eine Vielzahl von trivariablen Farbmodellen beschreiben.

[0004] Die Farbtransformation zwischen Modellsystemen bei der digitalen Datenverarbeitung stellt den Originalausrüstungshersteller vor viele Probleme. Die Übertragung von Daten von einem System zu einem anderen System ist schwierig, weil die Beziehung zwischen den Systemen im allgemeinen nichtlinear ist. Daher ist ein wesentliches Problem das Beibehalten der Farbintegrität zwischen einem Originalbild von einem Eingabegerät (wie z. B. einem Farbscanner, einer CRT-Anzeige, einer Digitalkamera, einer Computersoftware/Firmware-Generation und dergleichen) und einer übersetzten Kopie in einem Ausgabegerät (wie z. B. einer CRT-Anzeige, einem Farblaserdrucker, einem Farbtintenstrahldrucker und dergleichen).

[0005] Beispielsweise möchten Computerkünstler die Fähigkeit zum Erzeugen eines Farbbildes auf einem Computervideo und einem Drucker, der die gleiche Farbe in der Druckkopie liefert. Oder ein Originalfarbfoto kann mit einem Scanner digitalisiert werden, die resultierenden Daten können für die Anzeige auf einem Videobildschirm transformiert werden oder als eine Druckkopie durch einen Laser-, Tintenstrahl- oder thermischen Übertragungsdrucker reproduziert werden. Wie es in den zitierten Referenzmaterialien erörtert wird, können Farben als Aufbereitungen der additiven Primärfarben Rot, Grün und Blau (RGB) oder der subtraktiven Primärfarben Cyan, Magenta, Gelb und Schwarz (CMYK) aufgebaut werden.

[0006] Eine Transformation kann es erfordern, von einem RGB-Farbraum, beispielsweise einem Computervideobildschirm, zu einem CMYK-Farbraum zu gehen, beispielsweise einer Laserdruckerdruckkopie. Eine Transformation von einem Farbraum zu einem anderen erfordert komplexe nichtlineare Berechnungen in mehreren Dimensionen. Einige Transformationsoperationen könnten durch Matrixmultiplikation erreicht werden.

[0007] Eine Schwierigkeit bei diesem Verfahren der Farbraumumwandlung ergibt sich jedoch von Unvollkommenheiten bei den Farbstoffen, Phosphoren und Tonern, die für die Erzeugung der Farben verwendet werden. Eine zusätzliche Komplikation ist, daß unterschiedliche Medientypen unterschiedliche Farbantworten beim Drucken mit den gleichen Farbmittelmischungen erzeugen. Als Folge liefert ein rein mathematisches Farbraumumwandlungsverfahren keine annehmbare Farbreproduktion.

[0008] Es wurde erkannt, daß bei der Farbraumumwandlung hervorragende Ergebnisse erzielt werden unter Verwendung eines Nachschlagtabellenschemas auf der Basis eines Satzes von empirisch abgeleiteten Werten. Typischerweise verwendet der RGB-Farbraum, der für Videoanzeigen verwendet wird, acht Bits, um jede

der Primärfarben Rot, Grün und Blau darzustellen. Daher sind 24 Bits erforderlich, um jedes Bildelement darzustellen. Mit dieser Auflösung würde der RGB-Farbraum aus 2^{24} oder 16.777.216 Farben bestehen. Das Durchführen einer Farbraumumwandlung von jedem dieser Punkte in dem RGB-Farbraum zum Erzeugen der vier CMYK-Farbraumkomponenten (zum Beibehalten der Schwarzfarbreinheit beim Drucken ist normalerweise ein getrenntes Schwarz vorgesehen, anstatt mit allen drei Cyan-, Magenta- und Gelbfarbmitteln zu drucken, um zu erzeugen, was im allgemeinen als Prozeßschwarz bekannt ist) würden eine Nachschlagtabelle mit 4×2^{24} oder 67.108.864 Datenbytes erfordern. Der empirische Aufbau einer Nachschlagtabelle mit dieser Anzahl von Einträgen ist zu aufwendig.

[0009] Beim Durchführen der Transformation von einem Farbraum zu einem anderen kann eine Anzahl von Interpolationsschemata verwendet werden, die auf dem Gebiet der Farbraumumwandlung bekannt sind. Verfahren zum Durchführen von Farbraumumwandlung unter Verwendung von trilinearer Interpolation, Prismainterpolation und vierflächiger Interpolation sind in dem veröffentlichten Artikel PERFORMING COLOR SPACE CONVERSIONS WITH THREE DIMENSIONAL LINEAR INTERPOLATION, JOURNAL OF ELECTRONIC IMAGING, Juli 1995, Bd. 4(3), offenbart. Das U.S.-Patent Nr. 3,893,166, erteilt an Pugsley, offenbart ein Schema für eine Translation zwischen Farbräumen, das eine Nachschlagtabelle verwendet, um auf Werte zuzugreifen, die in einer Interpolation verwendet werden.

[0010] Das Dokument US-A-5475510 offenbart ein Farbsignaltransformationsverfahren, bei dem ein Objekt in Tetraeder unterteilt ist, wobei jeder durch einen von acht Gitterpunkten verläuft, die den Hexaeder des interpolierten Objekts bilden, und Zuweisen der unterschiedlichen Kombinationen von Interpolationsdaten zu den Tetraedern in einer Eins-zu-Eins-Entsprechung.

[0011] Die Umwandlung großer Datenmengen zwischen Farbräumen, wie es für Farbdrucken erforderlich ist, ist eine zeitaufwendige Operation, die herkömmliche Verfahren der Interpolation verwendet. Die Verwendung der rechenmäßig intensiven herkömmlichen Verfahren der Interpolation für den Farbraumumwandlungsprozeß macht es schwierig, hohe Datendurchsatzraten zu erreichen. Es besteht ein Bedarf an einem Interpolationsverfahren und einer Interpolationsvorrichtung, die eine Reduktion bei den Berechnungen ermöglicht, die zum Durchführen einer Umwandlung zwischen Farbräumen erforderlich sind.

Zusammenfassung der Erfindung

[0012] Die Erfindung ist in den angehängten Ansprüchen 1 und 10 dargelegt.

Beschreibung der Zeichnungen

[0013] Ein tieferes Verständnis der Erfindung wird durch die Betrachtung der folgenden detaillierten Beschreibung in Verbindung mit den beiliegenden Zeichnungen erhalten.

[0014] **Fig.** 1a ist eine Darstellung von Ausgangsfarbraumwerten, die für die Interpolation in einem Würfelgitter verwendet werden. Die Scheitel von jedem der Würfel, die das Würfelgitter bilden, stellen Werte des Ausgangsfarbraums dar.

[0015] **Fig.** 1b ist eine graphische Darstellung eines Farbraumumwandlungsprozesses von einer Farbe, die in einer zylindrischen Koordinate ausgedrückt ist, zu einer Farbe, die in einer rechteckigen Koordinate ausgedrückt ist.

[0016] **Fig.** 2a bis 2d sind graphische Darstellungen der Auswahl eines einzelnen Unterwürfels unter Verwendung der entsprechenden Bits von den Bits niedrigerer Ordnung des Eingangsfarbraumwerts.

[0017] **Fig.** 3a bis 3h zeigen die drei möglichen Unterwürfel, die von einem Würfel unter Verwendung der entsprechenden Bits niedrigerer Ordnung des Eingangsfarbraumwerts ausgewählt werden.

[0018] **Fig.** 4 zeigt die Numerierung der Scheitel des Würfels zum Zweck des Auswählens des Unterwürfels, der das Ergebnis der Interpolation enthält, unter Verwendung der entsprechenden Bits der Bits niedrigerer Ordnung.

[0019] **Fig.** 5 ist eine graphische Darstellung des radialen Unterwürfelerzeugungsprozesses.

[0020] **Fig.** 6a bis 6e zeigen eine graphische Darstellung mehrerer Iterationen des Würfelunterteilungspro-

zesses, der bei der radialen Interpolation verwendet wird.

[0021] Fig. 7 zeigt eine Hardwareimplementierung eines radialen Interpolators.

[0022] Fig. 8 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer radialen Interpolation.

[0023] Fig. 9 ist eine schematische Darstellung der Berechnungen, die für die Erzeugung der Unterwürfel bei der beschnittenen radialen Interpolation erforderlich sind.

[0024] Fig. 10 zeigt eine Hardwareimplementierung einer beschnittenen radialen Interpolation.

[0025] Fig. 11 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer beschnittenen radialen Interpolation.

[0026] Fig. 12 ist ein Flußdiagramm hoher Ebene eines Verfahrens, das in Software implementiert ist, zum Durchführen einer beschnittenen radialen Interpolation.

[0027] Fig. 13 ist eine Darstellung der äußeren Grenzen eines CMY- und eines RGB-Farbraums, die die Farben an den äußeren Grenzen der Farbräume zeigt.

[0028] Fig. 14 ist eine graphische Darstellung der Erzeugung eines Unterwürfels von zwei Tetraedern.

[0029] Fig. 15 ist eine schematische Darstellung der Berechnungen, die für die Erzeugung der Unterwürfel bei der beschnittenen vierflächigen Interpolation erforderlich sind.

[0030] Fig. 16 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer vierflächigen Interpolation.

[0031] Fig. 17 zeigt eine Hardwareimplementierung einer vierflächigen Interpolation.

[0032] Fig. 18 zeigt eine Hardwareimplementierung einer beschnittenen vierflächigen Interpolation.

[0033] Fig. 19 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer beschnittenen vierflächigen Interpolation.

[0034] Fig. 20 ist ein Flußdiagramm hoher Ebene eines Verfahrens zum Implementieren in Software zum Durchführen einer beschnittenen vierflächigen Interpolation.

[0035] Fig. 21 ist eine schematische Darstellung einer allgemeinen radialen Interpolation und einer beschnittenen vierflächigen Interpolationsimplementierung.

[0036] Fig. 22 zeigt eine Hardwareimplementierung einer allgemeinen beschnittenen radialen und beschnittenen vierflächigen Interpolation.

[0037] Fig. 23a bis 23a zeigen eine graphische Darstellung eines nicht-symmetrischen Interpolationsprozesses.

[0038] Fig. 24 zeigt eine graphische Darstellung der Erzeugung eines Unterwürfels von einem Würfel unter Verwendung einer nichtsymmetrischen Unterwürfelerzeugung.

[0039] Fig. 25 ist eine schematische Darstellung des nichtsymmetrischen radialen Interpolationsprozesses.

[0040] Fig. 26 ist ein Flußdiagramm hoher Ebene eines Verfahrens zum Durchführen einer nichtsymmetrischen radialen Interpolation.

[0041] Fig. 27 ist ein Flußdiagramm hoher Ebene eines Verfahrens, das in Software implementiert ist, zum Durchführen einer nichtsymmetrischen beschnittenen radialen Interpolation.

[0042] Fig. 28 zeigt eine Hardwareimplementierung einer nichtsymmetrischen beschnittenen radialen Inter-

polation.

[0043] Fig. 29 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer nichtsymmetrischen radialen Interpolation.

[0044] Fig. 30 zeigt eine Hardwareimplementierung einer nichtsymmetrischen radialen Interpolation.

[0045] Fig. 31 ist eine schematische Darstellung des nichtsymmetrischen beschnittenen vierflächigen Interpolationsprozesses.

[0046] Fig. 32 zeigt ein Flußdiagramm hoher Ebene eines Verfahrens zum Durchführen einer nichtsymmetrischen beschnittenen vierflächigen Interpolation.

[0047] Fig. 33 ist ein Flußdiagramm hoher Ebene eines Verfahrens, das in Software implementiert ist, um eine nichtsymmetrische beschnittene vierflächige Interpolation durchzuführen.

[0048] Fig. 34 zeigt eine Hardwareimplementierung einer nichtsymmetrischen beschnittenen vierflächigen Interpolation.

[0049] Fig. 35 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Implementieren einer nichtsymmetrischen vierflächigen Interpolation.

[0050] Fig. 36 zeigt eine Hardwareimplementierung eines nichtsymmetrischen vierflächigen Interpolators.

[0051] Fig. 37 zeigt eine Hardwareimplementierung eines allgemeinen nichtsymmetrischen beschnittenen radialen und nichtsymmetrischen beschnittenen vierflächigen Interpolators.

[0052] Fig. 38 umfaßt eine C-Codeauflistung eines Verfahrens zum Implementieren einer beschnittenen radialen Interpolationssoftware.

[0053] Fig. 39 umfaßt eine VHDL-Auflistung, die zum Erzeugen einer Hardwareimplementierung einer beschnittenen radialen Interpolation verwendet wird.

[0054] Fig. 40 umfaßt eine C-Codeauflistung eines Verfahrens zum Implementieren einer beschnittenen vierflächigen Interpolation in Software.

[0055] Fig. 41 umfaßt eine VHDL-Auflistung, die zum Erzeugen einer Hardwareimplementierung einer beschnittenen vierflächigen Interpolation verwendet wird.

[0056] Fig. 42 umfaßt eine VHDL-Auflistung, die zum Erzeugen einer Hardwareimplementierung einer allgemeinen beschnittenen radialen Interpolation und einer beschnittenen vierflächigen Interpolation verwendet wird.

[0057] Fig. 43 umfaßt eine C-Codeauflistung eines Verfahrens zum Implementieren einer nichtsymmetrischen beschnittenen radialen Interpolation in Software.

[0058] Fig. 44 umfaßt eine VHDL-Auflistung, die zum Erzeugen einer Hardwareimplementierung einer nichtsymmetrischen beschnittenen radialen Interpolation verwendet wird.

[0059] Fig. 45 umfaßt eine C-Codeauflistung eines Verfahrens zum Implementieren einer nichtsymmetrischen beschnittenen vierflächigen Interpolation in Software.

[0060] Fig. 46 umfaßt eine VHDL-Auflistung, die zum Erzeugen einer Hardwareimplementierung einer nichtsymmetrischen beschnittenen vierflächigen Interpolation verwendet wird.

[0061] Fig. 47 umfaßt eine VHDL-Auflistung, die zum Erzeugen einer Hardwareimplementierung einer allgemeinen nichtsymmetrischen beschnittenen radialen und einer nichtsymmetrischen beschnittenen vierflächigen Interpolation verwendet wird.

[0062] Die vorliegende Erfindung ist nicht auf die hierin dargestellten spezifischen beispielhaften Ausführungsbeispiele beschränkt sind. Außerdem, obwohl mehrere Ausführungsbeispiele der Unterwürfelinterpolation in dem Zusammenhang eines Farblaserdruckers erörtert werden, wird ein Durchschnittsfachmann auf diesem Gebiet nach dem Verstehen dieser Beschreibung erkennen, daß die offenbarten Ausführungsbeispiele der Unterwürfelinterpolation bei jeder interpolativen Datentransformation zwischen Räumen anwendbar sind. Beispielsweise könnten die Interpolationen, die für das Aufbereiten dreidimensionaler Graphiken erforderlich sind, die offenbarten Interpolationstechniken vorteilhaft verwenden.

[0063] Eine Unterwürfelinterpolation, die eine vierflächige Interpolation verwendet, um jeden der Scheitelpunkte der aufeinanderfolgenden Unterwürfel zu erzeugen, wird in dem Dokument US-A-5748176 gelehrt. Das Verfahren für die Erzeugung der Unterwürfelscheitelwerte, das in dieser mitanhängigen Anmeldung offenbart ist, erfordert jedoch eine große Anzahl von Berechnungen. Es besteht ein Bedarf nach einem Verfahren zum Erzeugen der Unterwürfelscheitelwerte, das rechentechnisch effizienter ist.

[0064] In **Fig.** 1a ist ein Würfelgitter 1 gezeigt. Das Würfelgitter ist aus einer Mehrzahl von Würfeln gebildet, wobei die Scheitel der Würfel Werte in dem Ausgangsfarbraum darstellen. Die Eingangsfarbraumwerte sind jeweils in einen oberen Abschnitt und einen unteren Abschnitt unterteilt. Der obere Abschnitt von jedem der Eingangsfarbraumwerte dient als ein Index zum Adressieren der Scheitelwerte des Würfelgitters 1, der für Interpolation verwendet wird. Der untere Teil von jedem der Eingangsfarbraumwerte wird verwendet, um zwischen den Ausgangsfarbraumwerten zu interpolieren, auf die unter Verwendung des unteren Teils des Eingangsfarbraumwerts zugegriffen wird. Jede der Abmessungen des Würfelgitters 1 entspricht einer der Komponenten des Eingangsfarbraumwerts. Die Werte, die den Scheiteln des Würfelgitters 1 zugeordnet sind, werden verwendet, um Ausgangsfarbraumwerte zu erzeugen.

[0065] Jeder der Aungangsfarbraumwerte hat mehrere Komponenten, die den Abmessungen des Ausgangsfarbraums entsprechen. Die Umwandlung wird von Eingangsfarbraumwerten zu Komponenten der Ausgangsfarbraumwerte durchgeführt. Die Umwandlung zu jeder Ausgangsfarbraumwertkomponente verwendet einen bestimmten Satz von Scheitelwerten. Für den Fall, in dem es drei Komponenten für jeden der Ausgangsfarbraumwerte gibt, gibt es drei Sätze von Scheitelwerten, die für die Farbraumumwandlung verwendet werden. Für diesen Fall wäre es möglich, jeden Scheitelwert als aus drei Werten gebildet anzusehen, wobei jeder der drei Werte von einem der drei Sätze ausgewählt ist. Wenn die Scheitelwerte auf diese Weise betrachtet werden, würde die Umwandlung zu jeder der Komponenten der Ausgangsfarbraumwerte parallel durchgeführt werden. Es ist auch möglich, die Umwandlung zu jeder der Ausgangsfarbraumkomponenten seriell durchzuführen. Wenn dieselbe auf diese Weise durchgeführt wird, kann die Umwandlung so betrachtet werden, daß sie drei getrennte Würfelgitter verwendet, wobei eines jedem Satz von Scheitelwerten entspricht.

[0066] In **Fig.** 1b ist eine allgemeine graphische Darstellung des Interpolationsprozesses gezeigt. Man betrachte beispielsweise die Umwandlung eines Eingangsfarbraumwerts (a, b, c) **10**, der eine Farbe in einem zylindrischen Farbraum darstellt, zu einem Ausgangsfarbraumwert (x, y, z) **11**, der die gleiche Farbe in einem kartesischen Farbraum darstellt. Bei diesem Beispiel sind sowohl a, b als auch c durch acht Bits dargestellt. Jede der drei Gruppen von acht Bits kann unterteilt werden, beispielsweise in vier obere Bits **10a** (dargestellt durch a_u, b_u und c_u) und vier untere Bits **10b** (dargestellt durch a_l, b_l und c_l). Die drei Gruppen von vier oberen Bits **10a** werden als ein Index in das Würfelgitter 1 verwendet, um die acht Werte wiederzugewinnen, die den Scheiteln eines Würfels in dem Würfelgitter 1 entsprechen, das als Interpolationsdatenwerte verwendet werden. Die drei Gruppen von vier unteren Bits **10b** werden dann verwendet, um zwischen den acht Interpolationsdatenwerten, die den Scheiteln des Würfelgitters 1 entsprechen, zu interpolieren, um eine Komponente des Ausgangsfarbraumwerts **11** zu erzeugen.

[0067] Ein Durchschnittsfachmann auf diesem Gebiet wird erkennen, daß andere Unterteilungen der Bits des Eingangsfarbraumwerts **10** möglich sind. Die spezielle Unterteilung der Bits hängt von Dingen ab, wie z. B. der Speichergröße, die verfügbar ist, um die Werte des Ausgangsfarbraums zu speichern, der für die Interpolation verwendet wird, und der Änderungsbetrag bei dem Ausgangsfarbraumwert, der zwischen Scheiteln des Würfelgitters 1 erscheint. Es gibt einen Kompromiß zwischen der Genauigkeit der Interpolation und der Speichergröße, die verwendet wird, um die Ausgangsfarbraumwerte zu speichern, die als die Interpolationsdatenwerte verwendet werden. Falls die Charakteristika des Ausgangsfarbraums derart sind, daß derselbe sich relativ linear durch den Farbraum ändert, dann sind weniger Scheitel in dem Würfelgitter 1 notwendig, um einen annehmbaren Interpolationsgenauigkeitspegel zu liefern.

[0068] Der Index, der durch a_u, b_u und c_u gebildet wird, dient als ein Eintrittspunkt in das Würfelgitter. Der Index adressiert einen Scheitel der acht Scheitel des Würfels, die als die Interpolationsdatenwerte verwendet werden. Jeder der Scheitel des Würfels entspricht einem Wert, der zum Interpolieren verwendet wird, um eine Komponente des Ausgangsfarbraumwerts **11** zu erzeugen. Die acht zugeordneten Scheitel eines Würfels in dem Würfelgitter 1 haben die folgenden relativen Adressen:
(a_u, b_u, c_u)
$(a_u + 1, b_u, c_u)$
$(a_u, b_u + 1, c_u)$
$(a_u, b_u, c_u + 1)$
$(a_u + 1, b_u + 1, c_u)$
$(a_u + 1, b_u, c_u + 1)$
$(a_u, b_u + 1, c_u + 1)$
$(a_u + 1, b_u + 1, c_u + 1)$

[0069] Das Wurfelunterteilungsinterpolationsverfahren, das in dem Dokument US-A-5748176 offenbart ist, führt eine Interpolation durch Erzeugen eines Unterwürfels unter Verwendung der Werte durch, die den Scheiteln des vorher erzeugten Unterwürfels zugeordnet sind. Der Anfangswürfel, der durch die Scheitelwerte gebildet wird, die den drei Gruppen von Bits höherer Ordnung (a_u, b_u, c_u) **10a** zugeordnet sind, wird verwendet, um den ersten Unterwürfel zu erzeugen. Dieser Anfangswürfel kann in acht Unterwürfel unterteilt werden. Diese drei Gruppen von Bits niedrigerer Ordnung **10b** (a_l, b_l, c_l) werden verwendet, um einen der acht möglichen Unterwürfel auszuwählen, die für die nächste Iteration der Unterwürfeldivision gebildet werden. Diese drei Gruppen von Bits niedrigerer Ordnung **10b** identifizieren, in welchem der acht möglichen Unterwürfel das Ergebnis der Interpolation positioniert sein wird. Wenn der Unterwürfel, der das Ergebnis der Interpolation enthält, identifiziert ist, wird dieser Unterwürfel verwendet, um den nächsten Unterwürfel zu erzeugen, der das Ergebnis der Interpolation enthält. Dieser Prozeß wird nachfolgend wiederholt, bis der letzte Unterwürfel, der das Ergebnis der Interpolation enthält, erzeugt wird. Einer der Werte, der einem Scheitel dieses letzten erzeugten Unterwürfels zugeordnet ist, wird als das Ergebnis der Interpolation verwendet.

[0070] **Fig.** 2a bis 2d stellen graphisch die Auswahl eines Unterwürfels dar, unter Verwendung der drei Gruppen von Bits niedrigerer Ordnung **10b** (a_l, b_l, c_l). Zum Zweck der Erklärung der Unterwürfelauswahl betrachte man den Fall, in dem die Bits niedrigerer Ordnung **10b** für jede Komponente des Eingangsfarbraumwerts aus vier Bits bestehen. In jeder der **Fig.** 2a bis 2d sind die Achsen gezeigt, die der a-, b- und c-Komponente des Eingangsfarbraumwerts entsprechen. Jede dieser Achsen entspricht einer Abmessung des Eingangsfarbraums. Unterwürfel werden unter Verwendung eines entsprechenden Bits (entsprechend in dem Sinne, daß dieselben Koeffizienten der gleichen Potenz von 2 sind) von den Bits niedrigerer Ordnung **10b** von jeder Komponente des Eingangsfarbraumwerts bestimmt. Jede Bitposition der Bits niedrigerer Ordnung **10b** von jeder Komponente kann so gesehen werden, daß sie den Würfel entlang der Abmessung, die dieser Komponente entspricht, halbiert. Der Wert des Bits für jede Komponente bestimmt, welche Hälfte des Würfels in der entsprechenden Abmessung ausgewählt wird, zum Zweck der Bestimmung, in welchem Unterwürfel das Ergebnis der Interpolation positioniert ist. Der ausgewählte Unterwürfel ist das Volumen, definiert durch die Schnittstelle der Würfelhälften, die durch die entsprechenden Bits jeder Komponente der Bits niedrigerer Ordnung **10b** des Eingangsfarbraumwerts ausgewählt werden. Falls das Bit der Bits niedrigerer Ordnung **10b** für die Komponente eine „1" ist, wird eine Ecke der ausgewählten Würfelhälfte entlang der entsprechenden Achse von dem Ursprung des Würfels um die Hälfte der Länge verschoben. Falls das Bit der Komponente eine „0" ist, umfaßt die Ecke der ausgewählten Würfelhälfte den Ursprung des Würfels. In **Fig.** 3a bis 3h sind die acht möglichen Unterwürfel gezeigt, die durch die gemeinsame Schnittstelle der Würfelhälften definiert sind, die durch das entsprechende Bit von a_l, b_l und c_l bestimmt sind.

[0071] Durch Numerieren der Scheitel der Würfel auf eine Weise, die der Zuweisung der Gruppen von Bits niedrigerer Ordnung **10b** der Komponenten der Eingangsfarbraumwerte zu den Achsen entspricht, wird der Scheitel des Würfels, der zum Erzeugen des Unterwürfels verwendet wird, der einem Scheitel des Unterwürfels zugeordnet ist, der erzeugt werden soll, durch den binären Wert bestimmt, der durch Kombinieren der entsprechenden Bits von jeder der Gruppen von Bits niedrigerer Ordnung **10b** der Komponenten gebildet wird. In **Fig.** 4 ist ein Würfel gezeigt, mit den Achsen gekennzeichnet und mit den Scheiteln numeriert. Der Würfel, der zum Erzeugen eines Unterwürfels verwendet wird, und der erzeugte Unterwürfel haben einen Scheitel gemeinsam. Mit dieser Zuweisung der Scheitelnummern ist die Nummer des Scheitels des Würfels, der zum Erzeugen des Unterwürfels verwendet wird, der in dem erzeugten Unterwürfel enthalten ist, der binäre Wert, der von den entsprechenden Bits von a_l, b_l und c_l für eine gegebene Bitposition gebildet wird.

[0072] Ein Beispiel wird erklärt, um die Interpolation darzustellen, die die Unterwürfelerzeugung verwendet.

Angenommen, die folgenden Werte werden für a_l, b_l und c_l verwendet:

Tabelle 1

$a_l = 1010$
$b_l = 1110$
$c_l = 0011$

[0073] Wenn diese Werte a_l, b_l und c_l zugewiesen sind, ist der Scheitel Nummer 6 (berechnet durch Auswählen des höchstwertigsten Bits von a_l, b_l und c_l und Verketten derselben in einen binären Wert) des Würfels, der zum Erzeugen des ersten Unterwürfels verwendet wird, auch ein Scheitel des ersten Unterwürfels. Der Scheitel des ersten Unterwürfels, der in dem erzeugten zweiten Unterwürfel enthalten ist, ist der Scheitel Nummer 2 (berechnet durch Auswählen des zweithöchstwertigsten Bits von jeweils a_l, b_l und c_l und Verketten derselben in einen binären Wert). Der Scheitel des zweiten Unterwürfels, der in dem dritten erzeugten Unterwürfel enthalten ist, ist Scheitel Nummer 7 (berechnet durch Auswählen des dritthöchstwertigsten Bits von jeweils a_l, b_l und c_l und Verketten derselben in einen binären Wert). Der Scheitel des dritten Unterwürfels, der in dem vierten erzeugten Unterwürfel enthalten ist, ist Scheitel Nummer 1 (berechnet durch Auswählen des vierthöchstwertigsten Bits von jeweils a_l, b_l und c_l und Verketten derselben in einen binären Wert). Die Scheitelwerte des ersten Unterwürfels werden mit den Scheitelwerten erzeugt, auf die unter Verwendung der Bits höherer Ordnung (a_u, b_u, c_u) **10a** der Komponenten des Eingangsfarbraumwerts zugegriffen wird. Die Scheitelwerte des zweiten Unterwürfels werden unter Verwendung der Scheitelwerte erzeugt, die für den ersten Unterwürfel erzeugt werden. Die Scheitelwerte des dritten Unterwürfels werden unter Verwendung der Scheitelwerte erzeugt, die für den zweiten Unterwürfel erzeugt werden.

[0074] Schließlich werden die Scheitelwerte des vierten Unterwürfels unter Verwendung der Scheitelwerte erzeugt, die für den dritten Unterwürfel erzeugt werden. Bei der Unterwürfelinterpolation ist der Wert, der dem Scheitel Nummer 0 des letzten erzeugten Unterwürfels zugeordnet ist, der Wert, der als das Ergebnis der Interpolation verwendet wird. Dieses Ergebnis ist eine Komponente des Ausgangsfarbraumwerts. Diese Unterwürfelerzeugungsprozedur könnte mit einer beliebigen Anzahl von Bits angewendet werden, die verwendet werden, um jede Komponente der Bits niedrigerer Ordnung **10b** des Eingangsfarbraumwerts zu spezifizieren.

[0075] Bisher wurden eine Vielzahl von Verfahren für die Erzeugung der Unterwürfelwerte verwendet. Diese Verfahren umfassen Tetraeder, Pyramide, Prisma und trilinear. Die radiale Unterwürfelerzeugung ist ein neues Verfahren der Unterwürfelerzeugung, das eine wesentliche Reduktion bei der Rechenkomplexität erreicht, die erforderlich ist, um die Unterwürfel zu erzeugen. Es sollte klar sein, daß jedes Interpolationsverfahren unterschiedliche Ergebnisse erzeugen kann, weil der Interpolationsprozeß eine Näherung der Farbraumumwandlung ist. Abhängig von der Position in dem Farbraum, wo die Umwandlung durchgeführt wird, und den bevorzugten Charakteristika des Ergebnisses, kann ein Verfahren wünschenswertere Ergebnisse erzielen als ein anderes.

[0076] In **Fig.** 5 ist eine graphische Darstellung des radialen Unterwürfelerzeugungsverfahrens gezeigt. Das deutliche Erklären des radialen Unterwürfelerzeugungsprozesses erfordert eine gewisse Notationsdefinition. Wie es vorher der Fall war, bezeichnen a_l, b_l und c_l die Bits niedrigerer Ordnung **10b** der jeweiligen Komponenten a, b und c des Eingangsfarbraumwerts. Der Wert der Variable i wird verwendet, um die Bitposition in den Bits niedrigerer Ordnung **10b** (a_l, b_l, c_l) zu bestimmen, wie es nachfolgend gezeigt ist, für den Fall, in dem vier Bits verwendet werden, um jede Komponente der Bits niedrigerer Ordnung **10b** zu bestimmen.

[0077] Der maximale Wert von i (ein Wert von 3) entspricht der höchstwertigsten Bitposition der Bits niedrigerer Ordnung **10b**. Der minimale Wert von i (ein Wert von 0) entspricht der niedrigstwertigsten Bitposition der Bits niedrigerer Ordnung **10b**. Wie es für einen Durchschnittsfachmann auf diesem Gebiet offensichtlich ist, kann diese Notation ohne weiteres an eine unterschiedliche Anzahl von Bits angepaßt werden, die für jede Komponente der Bits niedrigerer Ordnung **10b** verwendet wird. Für n Bits, die verwendet werden, um die Bits niedrigerer Ordnung **10b** darzustellen, reicht der Wert von i von n – 1 bis 0.

i:	3	2	1	0
a_l:	1	0	1	0
b_l:	1	1	1	0
c_l:	0	0	1	1

Tabelle 2

[0078] Unter Verwendung dieser Notation zeigt der Wert von i indirekt die Iteration der Unterwürfelerzeugung an. Ein Wert von i = 3 entspricht der Erzeugung des ersten Unterwürfels. Der erste Unterwürfel umfaßt den Scheitel Nummer 6 von dem Würfel, der durch Zugreifen auf die Werte des Würfelgitters 1 unter Verwendung der Bits höherer Ordnung **10a** gebildet wird. Für den Wert von i = 0 wird der vierte Unterwürfel erzeugt. Dieser vierte Unterwürfel umfaßt den Scheitel Nummer 1 von dem dritten erzeugten Unterwürfel. Um die Scheitelnummer des Würfels zu bestimmen, der zum Erzeugen des Unterwürfels verwendet wird, der in dem Unterwürfel enthalten ist, wird die folgende Gleichung verwendet:

Gl. 1a $$v(i) = 4a_l(i) + 2b_l(i) + c_l(i)$$

[0079] In der Gleichung 1a stellt v(i) die Scheitelnummer des Würfels dar, der in dem erzeugten Unterwürfel enthalten ist. $a_l(i)$, $b_l(i)$ und $c_l(i)$ stellen jeweils den binären Wert dar, der der „i-ten" Position in der jeweiligen Komponente der Bits niedrigerer Ordnung **10b** zugeordnet ist. Für jeden Wert von i ergibt die Gleichung 1a die korrekte Nummer des Scheitels des Würfels, der verwendet wird, um den Unterwürfel zu erzeugen, der in dem gewünschten Unterwürfel enthalten sein wird. Die Werte, die i annehmen kann, umfassen die Ganzzahl von n – 1 bis einschließlich 0, wobei n die Anzahl von Bits ist, die verwendet werden, um jede der Komponenten der Bits niedrigerer Ordnung **10b** des Eingangsfarbraumwerts zu spezifizieren. Der Wert des Scheitels mit der Nummer v(i) wird durch P[v(i)] bestimmt. Die Gleichung 1a kann für den Eingangsfarbraumwert **10** verallgemeinert werden, der aus d Komponenten gebildet ist. Nachfolgend ist ein verallgemeinerter Ausdruck für v[i] gegeben:

Gl. 1 $$bv[i] = 2^{d-1} \times lb_1[i] + 2^{d-2} \times lb_2[i] + 2^{d-3} \times lb_3[i] + ... + 2^{d-d} \times lb_d[i]$$

[0080] Bei der Gleichung 1b stellen jedes der „1b" die Bits niedrigerer Ordnung **10b** von einer der d Komponenten des Eingangsfarbraumwerts **10** dar. Wie bei der Gleichung 1a umfassen die Werte von i die Ganzzahlen von n – 1 bis einschließlich 0.

[0081] Die Werte, die den acht Scheiteln des Unterwürfels zugeordnet sind, werden von dem Würfel erzeugt, wie es in Tabelle 3 gezeigt ist. Diese Scheitelwerte, die durch P' bezeichnet sind [Unterwürfelscheitelnummer], stellen Unterwürfelscheitelwerte dar, und die Scheitelwerte, die durch P bezeichnet sind [Würfelscheitelnummer], stellen die Scheitelwerte des Würfels dar, die verwendet werden, um die Unterwürfelscheitelwerte zu erzeugen. Ein gegebener Wert eines Unterwürfelscheitels wird erzeugt durch Mitteln des entsprechenden Scheitelwerts des Würfels, von dem der Unterwürfel erzeugt wird, mit dem Wert des Scheitels des Würfels, der verwendet wird, um den Unterwürfel zu erzeugen, der in dem Unterwürfel enthalten ist (dieser Wert ist mit P[v(i)] bezeichnet).

Tabelle 3

P'[7] = (P[7] + P[v(i)]) ÷ 2
P'[6] = (P[6] + P[v(i)]) ÷ 2
P'[5] = (P[5] + P[v(i)]) ÷ 2
P'[4] = (P[4] + P[v(i)]) ÷ 2
P'[3] = (P[3] + P[v(i)]) ÷ 2
P'[2] = (P[2] + P[v(i)]) ÷ 2
P'[1] = (P[1] + P[v(i)]) ÷ 2
P'[0] = (P[0] + P[v(i)]) ÷ 2

[0082] In **Fig.** 6a bis 6e ist eine graphische Darstellung mehrerer Iterationen des Unterwürfeldivisionsprozesses gezeigt, der bei der radialen Interpolation verwendet wird. Die Werte, die für die Komponenten der Bits

niedrigerer Ordnung **10b** des Eingangsfarbraumwerts in dem Beispiel von **Fig.** 6 verwendet werden, sind die gleichen wie diejenigen, die in Tabelle 2 gezeigt sind. Die Werte der Scheitel des Würfels, der für die Erzeugung des ersten Unterwürfels verwendet wird (auf diese Werte wird unter Verwendung der Bits höherer Ordnung **10a** des Eingangsfarbraumwerts zugegriffen), werden von einer Farbtabelle geladen, die in dem Speicher gespeichert ist. Nach der letzten Iteration der Unterwürfeldivision wird der Scheitel Nummer 0 des letzten Unterwürfels als das Ergebnis des Interpolationsprozesses verwendet. Um die Akkumulation von Rundungsfehlern während der tatsächlichen Berechnung der Unterwürfelscheitelwerte zu verhindern, wird die erforderliche Division durch 2 für jede Iteration der Unterwürfelerzeugung nur an den Werten der Scheitel des erzeugten letzen Unterwürfels durchgeführt. Wenn die Division auf diese Weise durchgeführt wird, ist der verwendete Divisor 2^n, wobei n die Anzahl von Bits ist, die jeder Komponente der Bits niedrigerer Ordnung **10b** zugewiesen sind. Eine Division durch 2^n kann ohne weiteres durchgeführt werden durch Durchführen einer Verschiebeoperation nach rechts. Für den Fall, in dem n = 4 ist, ist dieser Divisor **16**. Wenn die Divisionsoperation bis nach der Erzeugung der Scheitelwerte des letzen Unterwürfels nicht durchgeführt wird, reduziert sich der Unterwürfelgenerationsprozeß auf eine Reihe von Additionen von ausgewählten Scheitelwerten der erzeugten Unterwürfel.

[0083] In **Fig.** 7 ist eine Hardwareimplementierung eines radialen Interpolators **100** gezeigt. Außerdem stellt **Fig.** 7 den Fortschritt der radialen Interpolation durch den radialen Interpolator **100** unter Verwendung von Werten für v[i] dar, die den Bits niedrigerer Ordnung **10b** der Tabelle 2 entsprechen. Jeder der Werte (P[0] bis P[7]), der den acht Scheiteln zugeordnet ist, die unter Verwendung der Bits höherer Ordnung **10a** ausgewählt wurde, ist zu einem Multiplexereingang des ersten Multiplexers **101** gekoppelt. Der Wert von v[i] für i = 3 ist zu dem Steuereingangssignal des ersten Multiplexers **101** gekoppelt. Der Wert von v[i] für i = 3 wird verwendet, um den Wert auszuwählen, der dem Scheitel des Würfels zugeordnet ist, der unter Verwendung der Bits höherer Ordnung **10a** ausgewählt wurde, die in dem ersten erzeugten Unterwürfel enthalten sind. Das Ausgangssignal des ersten Multiplexers **101** ist zu einem ersten Eingangssignal jedes Addierers eines ersten Satzes von Addierern **102** gekoppelt, der aus acht Addierern besteht. Das zweite Eingangssignal jedes Addierers des ersten Satzes von Addierern **102** ist zu einem der Werte gekoppelt, die unter Verwendung der Bits höherer Ordnung **10a** ausgewählt wurden. Der erste Multiplexer **101** und der erste Satz von Addierern **102** bilden eine erste Stufe des radialen Interpolators **100**. Es ist ersichtlich, daß bei dieser Konfiguration des Multiplexers **101** und des ersten Satzes von Addierern **102** die Mittelwertbildungsoperationen von Tabelle 3 (ohne die Division durch 2, die, wie vorher erwähnt wurde, verzögert wird, bis alle Iterationen der radialen Interpolation abgeschlossen sind) für eine einzige Iteration der radialen Interpolation abgeschlossen sind. Eine zweite, dritte und vierte Stufe des radialen Interpolators **100** sind jeweils aus einem zweiten Multiplexer **103** und einem zweiten Satz von Addierern **104**, einem dritten Multiplexer **105** und einem dritten Satz von Addierern **106** und einem vierten Multiplexer **107** und einem vierten Satz von Addierern **108** gebildet. Die Steuereingangssignale des zweiten **103**, dritten **105** und vierten **107** Multiplexereingangssignals sind jeweils gekoppelt zu v[i = 2], v[i = 1] und v[i = 0]. Die zweite, dritte und vierte Stufe des radialen Interpolators führen aufeinanderfolgende Iterationen der radialen Interpolation durch, wobei jede Iteration die Beziehungen von Tabelle 3 verwendet (erneut durch Verzögern der Division durch 2 bis zum Abschluß aller Iterationen).

[0084] In **Fig.** 8 ist ein Flußdiagramm hoher Ebene eines Verfahrens zum Durchführen einer einzigen Iteration einer radialen Interpolation gezeigt. Zunächst werden 2^d Interpolationsdatenwerte unter Verwendung Bits höherer Ordnung **10a** ausgewählt **200**. Für den Fall, in dem die radiale Interpolation für eine Farbraumumwandlung verwendet wird, entsprechen die Interpolationsdatenwerte den Scheitelwerten des ausgewählten Würfels. Außerdem ist für eine Farbraumumwandlung d typischerweise gleich 3, der Anzahl von Komponenten des Eingangsfarbraumwerts **10**. Nachdem Scheitelwerte ausgewählt sind **200**, wird die Scheitelnummer des Scheitelwerts, der für diese Iteration erforderlich ist, berechnet **201**. Für d = 3 wird die Gleichung 1a verwendet, um die erforderliche Scheitelnummer zu berechnen **201**. Abhängig von der Iteration der radialen Interpolation kann der Wert von i, der verwendet wird, um v[i] zu berechnen, von i = 3 bis i = 0 reichen. Nach der Berechnung **201** der Scheitelnummer wird einer der 2^d Interpolationsdatenwerte unter Verwendung der berechneten Scheitelnummer ausgewählt **202**. Schließlich wird ein Satz von 2^d Mittelwerten gemäß den Beziehungen von Tabelle 3 berechnet **203**. Um Rundungsfehler zu vermeiden, werden die erforderlichen Divisionen durch 2 zum Mittelwertbilden verzögert, bis alle Iterationen der Interpolation durchgeführt sind.

[0085] Es wurde erkannt, daß die Anzahl von Berechnungen, die erforderlich sind, um eine radiale Interpolation durchzuführen, wie es in **Fig.** 7 gezeigt ist, wesentlich reduziert werden könnten. Die Untersuchung des radialen Interpolationsprozesses von **Fig.** 7 offenbart, daß die Bestimmung des Interpolationsergebnisses nicht die Verwendung aller acht Scheitelwerte erfordert, auf die durch die Bits höherer Ordnung **10a** zugegriffen wird, noch die Verwendung aller in **Fig.** 7 gezeigten Addierer erfordert. In **Fig.** 9 ist eine schematische Darstellung der beschnittenen radialen Interpolation gezeigt. **Fig.** 9 ist verständlich durch Zurückarbeiten von dem in **Fig.** 7 gezeigten Interpolationsergebnis, um die Werte der Scheitel (auf die unter Verwendung der Bits höherer

Ordnung **10a** zugegriffen wird) zu bestimmen, die erforderlich sind, um das Ergebnis zu erzeugen. Wie es vorher erwähnt wurde, wird der Wert des Scheitels Nummer 0 (P[0]) des letzten Unterwürfels, der erzeugt wird, als das Ergebnis der Interpolation verwendet. Unter Verwendung der in Tabelle 3 aufgelisteten Gleichungen (ohne die Division durch 2) können die Scheitel des Unterwürfels unmittelbar vor dem letzen Unterwürfel, die verwendet werden, um den Wert der Scheitelnummer 0 des letzten Unterwürfels zu berechnen, bestimmt werden. Gleichartig dazu können die Gleichungen, die in Tabelle 3 aufgelistet sind, verwendet werden, um die Scheitel des Unterwürfels zwei vor dem letzen Unterwürfel zu bestimmen, die notwendig sind, um die benötigten Scheitel des Unterwürfels unmittelbar vor dem letzen Unterwürfel zu bestimmen.

[0086] Dieses Verfahren zum Bestimmen der Scheitel von jedem der Unterwürfel, die notwendig sind, um P[0] des letzen Unterwürfels zu berechnen, wird mit jedem Wert von i von 0 bis 3 für den Fall durchgeführt, in dem n = 4 ist. Wenn dies durchgeführt wird, zeigt das Ergebnis, daß die Werte, die verwendet werden, um P[0] des letzen Unterwürfels zu berechnen, aus nur einigen der Werten bestehen, die den Scheiteln des Würfels entsprechen, auf den durch die Bits höherer Ordnung **10a** des Eingangsfarbraumwerts zugegriffen wird. Als Folge werden nur **10** der **32** Addierer von **Fig.** 7 zum Berechnen eines Interpolationsergebnisses von einem gegebenen Eingangsfarbraumwert **10** verwendet. Für die Werte der Bits niedrigerer Ordnung **10b**, die in Tabelle 2 gezeigt sind, sind die Scheitelnummern des Würfels, auf die durch die Bits höherer Ordnung **10a** zugegriffen wird, denen die Werte entsprechen, die verwendet werden, um P[0] des letzen Unterwürfels zu berechnen: 0, 6, 2, 7 und 1. Die Werte der Scheitel des Würfels, auf die durch die Bits höherer Ordnung **10b** zugegriffen wird, die dem entsprechen, sind: P[0], P[6], P[2], P[7] und P[1]. Im allgemeinen sind die Werte der Scheitel des Würfels, auf die unter Verwendung der Bits höherer Ordnung **10a** zugegriffen wird, die verwendet werden, um P[0] des letzten Unterwürfels zu berechnen: P[0], P[v(i = 3)], P[v(i = 2)], P[v(i = 1)], P[v(i = 0)]. Der allgemeine Ausdruck, der für n = 4 abgeleitet werden kann, ist:

Gl. 2 $$P[0]_{\text{letzter Unterwürfel}} = (\{8 \times P[v(i = 3)]\} + \{4 \times P[v(i = 2)]\} + \{2 \times P[v(i = 1)]\} + P[v(i = 0)] + P[0] \div 16$$

[0087] Die Gleichung 2 ist ein Ausdruck zum Berechnen eines Ergebnisses unter Verwendung einer beschnittenen radialen Interpolation mit n = 4. Ein verallgemeinerter Ausdruck für die beschnittene radiale Interpolation ist:

Gl. 3 $$P[0]_{\text{letzter Unterwürfel}} = \left\{\sum_{i=0}^{n-1}\left(2^i \times P[v(i)]\right) + P[0]\right\} \div 2^n$$

[0088] Die Gleichung 3 kann verwendet werden, um die Berechnung des beschnittenen radialen Interpolationsergebnisses zu verallgemeinern. Es sollte angemerkt werden, daß in Gleichung 2 und dem verallgemeinerten Ausdruck in Gleichung 3 der Wert, der der Scheitelnummer 0 des Würfels zugeordnet ist, der unter Verwendung der Bits höherer Ordnung **10a** ausgewählt wird, immer verwendet wird. Würde der Wert einer anderen Scheitelnummer als der Scheitelnummer 0 des letzen Unterwürfels, der erzeugt wurde, als das Ergebnis der Interpolation verwendet, würde der Wert dieser Scheitelnummer des ursprünglich ausgewählten Würfels statt P[0] verwendet werden.

[0089] Die Hardwarefunktionsblöcke, die erforderlich sind, um die beschnittene radiale Interpolation durchzuführen, umfassen Addierer und Multiplexer. Mit D Abmessungen in dem Ausgangsfarbraum und n Bits, die jede Gruppe von Bits niedrigerer Ordnung **10b** des Eingangsfarbraumwerts darstellen, können die Anforderungen der Hardwareimplementierung der beschnittenen radialen Interpolation wie folgt berechnet werden:

Gl. 4 $$\text{Anzahl von Addierern} = D \times (n + 1)$$

Gl. 5 $$\text{Anzahl von Multiplexern} = D \times n$$

[0090] Es sollte angemerkt werden, daß der zusätzliche Addierer, der in Gleichung 4 spezifiziert ist, zum Zweck des Rundens verwendet wird. Zusätzliche Operationen, die durch die Hardware durchgeführt werden müssen, umfassen Multiplikation, Division und Verkettung. Die Multiplikations- und Divisionsoperationen mit einer Potenz von 2 können durch Verschieben von Bitpositionen durchgeführt werden. In Hardware wird dieses Verschieben erreicht durch Verbinden einer Leitung, die einem Bit entspricht, mit einer Position höherer Ordnung für die Multiplikation oder mit einer Position niedrigerer Ordnung für die Division. In Hardware wird eine Verkettung erreicht durch Gruppieren von Leitungen, die Bitpositionen entsprechen. Daher können die Multiplikations-, Divisions- und Verkettungsoperationen durchgeführt werden, ohne daß zusätzliche Hardware hinzugefügt werden muß.

[0091] Um den Gatterpegelentwurf zu erzeugen, der notwendig ist, um die beschnittene radiale Interpolation in Hardware zu implementieren, kann eine üblicherweise verwendete Hardwarebeschreibungssprache, wie z. B. VHDL, verwendet werden. In **Fig.** 39 ist eine Auflistung des VHDL-Codes enthalten, der eine Hardwareimplementierung einer beschnittenen radialen Interpolation erzeugen kann.

[0092] In **Fig.** 10 ist eine Hardwareimplementierung eines beschnittenen radialen Interpolators **300** für n = 4 gezeigt. Es sollte klar sein, daß die in **Fig.** 10 gezeigte Hardwareimplementierung verwendet werden kann, um eine einzige Komponente des Ausgangsfarbraumwerts **11** zu erzeugen. Diese gleiche Hardware könnte wiederholt für zusätzliche (D – 1) Durchläufe verwendet werden, um die verbleibenden D – 1 Komponenten des Ausgangsfarbraumwerts **11** zu erzeugen. Oder es könnte zusätzliche (D – 1) Duplikationen der in **Fig.** 9 gezeigten Hardwareimplementierung geben, um jede der D Komponenten gleichzeitig zu erzeugen. Der beschnittene radiale Interpolator **300** ist eine Hardwareimplementierung der Gleichung 2 ohne die Division durch 16. Die Division durch 16 könnte durch Bitverschiebung des Ergebnisses der Additionen erreicht werden. Die Auswahl von vier der fünf Scheitelwerte (P[6], P[2], P[7] und P[1]), die verwendet werden, um das Interpolationsergebnis zu berechnen, erfordert vier Multiplexer **301**–**304**.

[0093] Die vier erforderlichen Additionen werden unter Verwendung von vier Addierern **305**–**308** erreicht. Der fünfte Scheitelwert, der für die Berechnung des Interpolationsergebnisses P[0] erforderlich ist, ist in die Eingänge von einem der Addierer fest verdrahtet. Es sollte angemerkt werden, daß aufgrund der assoziativen Eigenschaft der Addition die Hardwareimplementierung von **Fig.** 10 implementiert werden kann, so daß die Additionen in einer Anzahl von unterschiedlichen Reihenfolgen durchgeführt werden. Die in **Fig.** 1 gezeigte Reihenfolge minimiert die Ausbreitungsverzögerung durch den Addierer. Ferner können andere Einrichtungen zum Addieren verwendet werden. Beispielsweise könnte ein einzelner Addierer, der eine ausreichende Anzahl von Eingängen aufweist, verwendet werden. Oder ein Mikroprozessor könnte verwendet werden, um die Additionen zu erreichen.

[0094] Die drei Multiplikationsoperationen **309**–**311** entsprechen der Multiplikation mit den Koeffizienten 8, 4 und 2 der ersten drei Terme auf der rechten Seite der Gleichung 2. Es sollte angemerkt werden, daß die Multiplikationsoperationen **309**–**311** in Hardware erreicht werden durch Führen der Leitungen entsprechend den Bitpositionen auf jeden der entsprechenden Multiplexerausgangssignale. Daher werden diese Multiplikationen ohne zusätzliche Hardwarekosten implementiert. Es ist möglich, die Multiplikationen durch das Führen von Leitungen zu implementieren, weil alle Koeffizienten Potenzen von 2 sind.

[0095] Ein Durchschnittsfachmann auf diesem Gebiet erkennt, daß die in **Fig.** 10 gezeigte Hardwareimplementierung für Werte von n größer 4 oder kleiner als 4 angepaßt werden kann. Man betrachte die Hardwareimplementierung eines beschnittenen radialen Interpolators für n = 1. Diese Hardwareimplementierung einer beschnittenen radialen Interpolation wäre sinnvoll für eine Interpolation, die eine einzelne Iteration von Würfelunterteilung durchführt, und dann einen der Scheitelwerte des erzeugten Unterwürfels als das Interpolationsergebnis auswählt. Diese Hardwareimplementierung der beschnittenen radialen Interpolation erfordert nur einen einzigen Multiplexer und einen einzigen Addierer (selbstverständlich erfordert das Runden am Ende einen zusätzlichen Addierer, aber dieser zusätzliche Addierer ist in **Fig.** 10 nicht gezeigt).

[0096] In **Fig.** 11 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens für eine beschnittene radiale Interpolation für die Eingangsfarbraumwerte **10** mit d Komponenten mit jedem Satz von Bits niedrigerer Ordnung **10b** mit n Bits gezeigt. Zunächst werden n Werte unter Verwendung der Gleichung 1b berechnet **400**, danach werden n + 1 Interpolationsdatenwerte (die den Scheitelwerten in einer Farbraumumwandlung entsprechen) unter Verwendung der berechneten n Werte und Bits höherer Ordnung **10a** ausgewählt **401**. Schließlich wird das Interpolationsergebnis durch Multiplizieren und Addieren der ausgewählten n + 1 Interpolationsdatenwerte gemäß Gleichung 3 berechnet **402**.

[0097] Eine Softwareimplementierung der beschnittenen radialen Interpolation ist rechentechnisch sehr effizient. Mit d Eingangsabmessungen, D Ausgangsabmessungen und 2^n Werten zwischen Scheiteln des Würfelgitters 1 kann die Anzahl von Berechnungen, die erforderlich, sind, um ein Interpolationsergebnis zu erzeugen, berechnet werden als:

Gl. 6 $$\text{Anzahl von ALU-Operationen} = 2 \times n \times (d + D - 1) + D$$

[0098] Die Anzahl von Speicherzugriffen, die erforderlich sind, um das Interpolationsergebnis zu erzeugen, können berechnet werden als:

Gl. 7 Anzahl von Speicherzugriffen = D × (n + 1)

[0099] Es sollte angemerkt werden, daß anders als viele Interpolationsverfahren, sowohl die Anzahl der ALU-Operationen als auch die Anzahl von Speicherzugriffen bei D, d und n linear sind, was zu der relativen Recheneffizienz der beschnittenen radialen Interpolation führt.

[0100] In **Fig.** 12 ist ein Flußdiagramm hoher Ebene eines Verfahrens gezeigt, das in Software implementiert ist, um die beschnittene radiale Interpolation durchzuführen. Zunächst wird eine Bestimmung **500** durchgeführt, ob irgendeine der Komponenten des Eingangsfarbraumwerts (a, b, c) **10** einer Position an einer äußeren Grenze des Würfelgitters 1 entspricht. Dies ist der Fall, falls eine oder mehrere der Komponenten des Eingangsfarbraumwerts einen hexadezimalen Wert von FF aufweist. Falls dies der Fall ist, dann wird den Komponenten des Eingangsfarbraumwerts **10**, die einen hexadezimalen Wert von FF aufweisen, zum Zweck des Erzeugens des Index in das Würfelgitter 1 zum Wiedergewinnen der notwendigen Scheitelwerte ein hexadezimaler Wert von **100** zugewiesen **501**.

[0101] Das Zuweisen eines hexadezimalen Werts von **100** zu den hexadezimalen Eingangsfarbraumwerten von FF wird durchgeführt zum Adressieren eines speziellen Falles in der Interpolation. Um diesen speziellen Fall darzustellen, betrachte man die Darstellung der Eingangsfarbraumwerte **10** durch acht Bits für jede Komponente, wobei jede Komponente in vier Bits höherer Ordnung und vier Bits niedrigerer Ordnung unterteilt ist. Mit dieser Unterteilung können die Bits höherer Ordnung die hexadezimalen Indexwerte 00, 10, 20, 30, ... F0 für jede Komponente bilden. Die vier Bits niedrigerer Ordnung für jede Komponente werden verwendet, um zwischen den Ausgangsfarbraumwerten **11** zu interpolieren, auf die unter Verwendung der Indexwerte zugegriffen wird. Die Differenz zwischen dem Paar von Ausgangsfarbraumwerten **11**, auf die unter Verwendung aufeinanderfolgender hexadezimaler Indexwerte von 00 bis F0 zugegriffen wird, wird in 16 gleichen Inkrementen gespannt. Mit jedem aufeinanderfolgenden Inkrement wird der zugeordnete Wert um 1/16 der Differenz zwischen dem zugegriffenen Paar von Ausgangsfarbraumwerten **11** erhöht, wenn von dem niedrigen Ausgangsfarbraumwert zu dem höheren Ausgangsfarbraumwert gegangen wird. Beispielsweise ist der zugeordnete Wert nach fünf Inkrementen 5/16 der Differenz zwischen dem zugegriffenen Paar von Ausgangsfarbraumwerten **11**. Unter Verwendung der vier Bits niedrigerer Ordnung wird der Wert, der der entsprechenden Anzahl von Inkrementen zugeordnet ist, zu dem Ausgangsfarbraumwert **11** addiert, der unter Verwendung der Bits höherer Ordnung, zum Erzeugen des Interpolationsergebnisses ausgewählt wurde. Es ergibt sich jedoch ein Problem zwischen den Indexwerten F0 und FF (Indexwert **100** existiert nicht in der Tabelle) für jede Komponente des Eingangsfarbraumwerts **10**. Zwischen F0 und FF gibt es nur 15 Inkremente und der Ausgangsfarbraumwert **11**, auf den durch FF zugegriffen wird, entspricht einer äußeren Grenze des Ausgangsfarbraums. Der Interpolationsprozeß ist jedoch entworfen, um auf 16 Inkrementen zwischen den Ausgangsfarbraumwerten **11** zu arbeiten, auf die unter Verwendung der Indexwerte zugegriffen wird. Um dieses Problem zu adressieren, werden die Ausgangsfarbraumwerte **11**, die dem Indexwert FF entsprechen, auf eine Position abgebildet, die eine hexadezimale Adresse von **100** aufweist. Diese Abbildung verteilt die Differenz bei den Ausgangsfarbraumwerten **11**, die den hexadezimalen Indexwerten F0 und FF entsprechen, effektiv über 16 Inkremente anstatt 15. Deswegen wird es leichte Fehler geben, die sich von der Interpolation zwischen den Indexwerten F0 und FF ergeben. Obwohl dies in den Hardwareblockdiagrammen nicht gezeigt ist, wird die Handhabung dieses speziellen Falls in der Interpolation in den Hardwareimplementierungen der verschiedenen Interpolatorausführungsbeispiele durchgeführt.

[0102] Nach jeder notwendigen Neuzuweisung des Eingangsfarbraumwerts **10** werden die Indizes, die verwendet werden, um auf die Werte zuzugreifen, die den erforderlichen Scheiteln des ausgewählten Würfels in dem Würfelgitter 1 entsprechen, berechnet **502**. Schließlich werden die Werte für jede Komponente des Ausgangsfarbraumwerts (x, y, z) **11** berechnet **503**. In **Fig.** 38 dieser Beschreibung wird der Code einer Implementierung in C für n = 4 des Verfahrens hoher Ebene der in **Fig.** 12 gezeigten beschnittenen radialen Interpolation geliefert.

[0103] Es sollte klar sein, daß eine Anzahl möglicher prozessorspezifischen Optimierungen der Software zum Durchführen einer beschnittenen radialen Interpolation durchgeführt werden können. Beispielsweise kann durch Kombinieren aller Komponenten jedes Ausgangsfarbraumwerts **11** in ein einziges Wort die Anzahl von Speicherzugriffen, die erforderlich sind, um die Umwandlung zu dem Ausgangsfarbraumwert **11** durchzuführen, reduziert werden. Eine weitere mögliche Optimierung nutzt die Fähigkeit des ALU aus, 32-Bit-Operationen durchzuführen. Durch Zuweisen der Bits 0–7 eines ALU-Worts zum Handhaben der Berechnung der y-Komponente des Ausgangsfarbraumwerts und der Bits 16–23 zum Handhaben der Berechnung der x-Komponente des Ausgangsfarbraumwerts kann eine einzige Sequenz von Verschiebungen und Additionen verwendet werden, um die x- und y-Komponenten parallel zu erzeugen. Es ist auch möglich, eine beschnittene radiale Inter-

polation in Hardware zu implementieren. Die Recheneffizienzen, die in der Softwareimplementierung der beschnittenen radialen Interpolation vorlagen, bestehen in der Hardwareimplementierung als reduzierte Hardwareerfordernisse.

[0104] Die vierflächige Interpolation unterteilt den Würfel, auf den durch die Bits höherer Ordnung **10a** des Farbraumeingangswerts zugegriffen wird, in eine Anzahl von Tetraedern, die für die Erzeugung des Unterwürfels verwendet werden, der das Ergebnis der Interpolation enthält. Der resultierende Unterwürfel wird dann in Tetraeder unterteilt. Zwei dieser Tetraeder werden dann verwendet, um noch einen weiteren Unterwürfel zu erzeugen, der das Ergebnis der Interpolation enthält. Die nachfolgende Teilung der erzeugten Unterwürfel in Tetraeder wird n mal durchgeführt, wobei n die Anzahl von Bits ist, die verwendet wird, um jede der Komponenten der Bits niedrigerer Ordnung **10b** des Eingangsfarbraumwerts **10** darzustellen.

[0105] In **Fig.** 13 ist eine Darstellung der äußeren Grenzen eines CMY- oder eines RGB-Farbraums gezeigt. Wie es in **Fig.** 13 ersichtlich ist, umfassen die Scheitel des Würfels **600**, der durch die äußeren Grenzen dieser Farbräume gebildet wird, Werte, die den Bestandteilfarben von jedem dieser Farbräume entsprechen. Eine Charakteristik der CMY- und RGB-Farbräume ist, daß die Diagonale, die zwischen dem weißen **601** und dem schwarzen **602** Scheitel des Farbraums verbunden ist, der Luminanzachse entspricht. Punkte entlang der Luminanzachse haben Werte, die den verschiedenen Grauschattierungen entsprechen. Wie es vorher erwähnt wurde, werden die Bits höherer Ordnung **10a** des Eingangsfarbraumwerts **10** verwendet, um auf acht zugeordnete Werte zuzugreifen, die einen Würfel bilden, der in dem Würfel **600** positioniert ist. Analog zu dem Würfel **600**, der den CMY- oder RGB-Farbraum darstellt, kann jeder der ausgewählten Würfel als eine Art von Miniaturfarbraum angesehen werden, wobei die Werte jedem der acht Scheitel entsprechen, die Farben aufweisen, die zu den Farben der entsprechenden Scheitel des Würfels **600** gewichtet werden. Beispielsweise ist der Scheitel des ausgewählten Würfels, der räumlich dem gelben Scheitel **603** entspricht, der Scheitel, der einen Wert am nächsten zu dem Wert für die Farbe Gelb in dem ausgewählten Würfel hat. Die anderen sieben Scheitel des ausgewählten Würfels können auf ähnliche Weise betrachtet werden. Der diagonal verbindende Scheitel 0 und der Scheitel 7 dienen dazu, eine konstante Chromalinie zwischen den Farben zu definieren, die den Scheiteln des ausgewählten Würfels zugeordnet sind.

[0106] Von der Reproduktion von Farben in dem Druckprozeß können sich bestimmte Artefakte ergeben. Diese Artefakte sind visuell wahrnehmbar als Farbe, die von denjenigen abweichen, die durch den Farbraumwert, der in den Druckprozeß eingegeben wird, spezifiziert werden. Die Artefakte sind besonders bemerkenswert für Eingangsfarbraumwerte, die nahe der Luminanzachse positioniert sind. Eingangsfarbraumwerte nahe der Luminanzachse entsprechen Grauschattierungen mit kleinen Mengen an Farbe. Faktoren in dem Farbreproduktionsprozeß, die die resultierende Farbe weiter von der Luminanzachse verschieben können als beabsichtigt, werden in einem Graufeld ohne weiteres wahrgenommen. Die Artefakte erscheinen als Farben in den Feldern, die nur verschiedene Grauschattierungen entlang der Luminanzachse enthalten sollten. Die Artefakte können sich unter anderem von den Charakteristika des Prozesses ergeben, der zum Drucken verwendet wird (wie z. B. ein elektrophotographischer oder ein Tintenstrahldruckprozeß), oder von Charakteristika der Farbmittel (wie z. B. Toner oder Tinte), die beim dem Druckprozeß verwendet werden. Die Veränderbarkeit bei den Parametern des Druckprozesses führt zu der Reproduktion von Farben aus der Luminanzachse, wenn die Ergebnisse hätten grau sein sollen.

[0107] Die vierflächige Interpolation reduziert in einigen Fällen den Grad, zu dem diese Artefakttypen wahrnehmbar sind. Die Reduktion bei Druckartefakten erscheint, weil der Wert eines Scheitels des Unterwürfels, der von dem Tetraeder erzeugt wird, unter Verwendung der Werte berechnet wird, die dem Scheitel Nummer 0 und dem Scheitel Nummer 7 zugeordnet sind. Wie es vorher erwähnt wurde, definiert die Diagonale, die zwischen dem Scheitel Nummer 0 und dem Scheitel Nummer 7 gebildet ist, eine konstante Chromalinie für den ausgewählten Würfel. Das Berechnen eines Scheitels des Unterwürfels entlang dieser Mittelpunktfarbgrenzlinie erzeugt eine Gewichtung bei der Interpolation, die dazu neigt, die Änderungsrate bei dem Ausgangsfarbraumwert zu reduzieren, während sich der Eingangsfarbraumwert **10** von der Diagonale des Würfels weg bewegt, der durch die Bits höherer Ordnung **10a** ausgewählt ist. Dies wiederum neigt dazu, die Veränderbarkeit bei den Druckprozeßparametern auszugleichen, die eine Nichtgrauausgabe mit grauen Eingangsfarbraumwerten erzeugen.

[0108] In **Fig.** 14 ist eine graphische Darstellung der Erzeugung eines Unterwürfels **700** von einem Tetraeder **701** gezeigt. Jeder Wert eines Scheitels des Tetraeders **701**, der verwendet wird, um den Wert eines Scheitels des Unterwürfels **700** zu berechnen, ist auch ein Wert eines Scheitels des Würfels **702**, von dem der Tetraeder **701** abgeteilt wurde. Man lasse P[k] den Wert bezeichnen, der dem Scheitel k eines Würfels zugeordnet ist. Man lasse P'[k] den Wert bezeichnen, der dem Scheitel k eines Unterwürfels zugeordnet ist, der in dem Würfel

mit dem Scheitel k enthalten ist. Es kann gezeigt werden, daß der Wert P'[k] berechnet wird als:

Gl. 8 $$P'[k] = \{P[k \,\&\, v(i)] + P[k|v(i)]\} \div 2$$

„&" stellt die bitweise UND-Operation dar
„|" stellt die bitweise ODER-Operation dar
„k" stellt die Scheitelnummer dar

$v(i) = 4 \times a(i) + 2 \times b(i) + c(i)$

i stellt die Bitposition in den Bits niedrigerer Ordnung **10b** in dem Eingangsfarbraumwert dar.

[0109] Die Unterwürfelerzeugung, die unter Verwendung der Gleichung 8 erreicht werden kann, liefert ein neues Verfahren zum Berechnen von Scheitelwerten für eine vierflächige Interpolation. Durch Verwenden der Gleichung 8 können die Indizes berechnet werden, die verwendet werden, um auf die Werte der Scheitel, die zum Berechnen der Unterwürfelscheitelwerte verwendet werden, zuzugreifen. Dies liefert einen Vorteil im Vergleich zu Interpolationsverfahren, die das Zugreifen auf eine Nachschlagtabelle erfordern, um die Indizes zu bestimmen, die verwendet werden, um auf die Scheitelwerte zuzugreifen. Die Verwendung einer Nachschlagtabelle erfordert Speicherzugriffe. Folglich erfordert das Verwenden einer Nachschlagtabelle zum Erzeugen der Indizes eine wesentlich größere Anzahl von Maschinenzyklen als unter Verwendung des Prozessors zum Berechnen der Indizes erforderlich wären. Daher liefert das Verwenden der Gleichung 8 zum Berechnen der Indizes, die verwendet werden, um auf die Scheitelwerte zuzugreifen, einen wesentlichen Geschwindigkeitsvorteil bei der vierflächigen Interpolation im Vergleich zu bisherigen Verfahren zum Durchführen einer vierflächigen Interpolation. Ferner liefert das Implementieren der Gleichung 8 in Hardware zum Berechnen der Multiplexersteuereingangssignale, die verwendet werden, um die Scheitelwerte auszuwählen, eine einfachere Hardwareimplementierung von vierflächiger Interpolation.

[0110] Wie es bei der beschnittenen radialen Interpolation der Fall war, ist der Wert des Scheitels Nummer 0 des letzten erzeugten Unterwürfels das Ergebnis bei der vierflächigen Interpolation. Es wurde erkannt, daß nicht alle Pegel der Scheitel aller erzeugten Unterwürfel erforderlich waren, um das Ergebnis der Interpolation zu erzeugen. Dies führte zu der Entwicklung einer weiteren Verbesserung bei vierflächiger Interpolation, die als beschnittene vierflächige Interpolation bezeichnet wird. In **Fig.** 15 ist eine schematische Darstellung einer beschnittenen vierflächigen Interpolation gezeigt. In **Fig.** 15 zeigt der Hauptindikator, der dem Term zugeordnet ist, der den Wert jedes Scheitels darstellt, den Pegel der Unterwürfelerzeugung an. Beispielsweise stellen die Terme, die mit P'[] bezeichnet sind, Scheitelwerte nach der ersten Würfelunterteilungsiteration dar, die Terme, die als P''[] bezeichnet sind, stellen Scheitelwerte nach der zweiten Würfelunterteilungsiteration dar. Dieses Verfahren des Bestimmens von Scheitelwerten gilt für die Erzeugung von aufeinanderfolgenden Unterwürfeln. **Fig.** 15 stellt die beschnittene vierflächige Interpolation unter Verwendung von vier Bits für die Bits niedrigerer Ordnung **10b** des Eingangsfarbraumwerts **10** dar. Die in **Fig.** 15 gezeigten Terme können erzeugt werden durch Beginnen mit dem Endergebnis der Interpolation P''''[0] und, unter Verwendung der Gleichung 8, nachfolgendes Bestimmen der Werte der Scheitel des vorherigen Unterwürfels, die erforderlich sind, um die Werte der Scheitel des aktuellen Unterwürfels zu erzeugen, bis die Werte, die erforderlich sind, um den aktuellen Unterwürfel zu erzeugen, erhalten werden durch Zugreifen auf die Werte der Scheitel in dem Würfelgitter 1 unter Verwendung von Bits höherer Ordnung **10a**. Wie es für die beschnittene radiale Interpolation der Fall war, wird die Durch-2-Dividieren-Operation nicht durchgeführt, bis der Wert der Scheitel Nummer 0 des letzten Unterwürfels erhalten wird, um die Akkumulation von Rundungsfehlern zu verhindern.

[0111] Eine beschnittene vierflächige Interpolation liefert wesentliche Berechnungseinsparungen im Vergleich zu einer vierflächigen Interpolation. Mit d Eingangsabmessungen, D Ausgangsabmessungen und n Bits niedrigerer Ordnung **10b** wird die Anzahl von Berechnungen, die erforderlich ist, um die beschnittene vierflächige Interpolation durchzuführen, berechnet als:

Gl. 9
$$\text{Anzahl von ALU - Operationen} = n \times [(2 \times d) - 1] + 2 \times \left[\sum_{i=1}^{n-1} (2^i - 1) + D \times (2^n + 1)\right]$$

Gl. 10 $$\text{Anzahl von Speicherzugriffen} = D \times [\min(2^d, 2^n)]$$

[0112] Es sollte angemerkt werden, daß für diese Gleichungen die Berechnungen linear variieren als eine Funktion von d und D für die Anzahl von ALU-Operationen, exponentiell als eine Funktion von d für die Anzahl

von Speicherzugriffen und exponentiell als eine Funktion von n für die Anzahl von Speicherzugriffen. Wie es in **Fig.** 15 gezeigt ist, ist die beschnittene vierflächige Interpolation implementiert, so daß 2^n Speicherzugriffe erforderlich sind. Es ist jedoch möglich, eine beschnittene vierflächige Interpolation zu implementieren, so daß die Anzahl von Speicherbezugnahmen, die erforderlich ist, ein Maximum von 2^d ist, was für d = 3 die Anzahl von Scheiteln in einem Würfel ist. Dies wird durchgeführt durch Erkennen, daß in den 2^n Speicherzugriffen Redundanz existiert. Durch Verwenden einiger der zugegriffenen Scheitelwerte für mehrere der Eingangswerte, die in **Fig.** 15 erforderlich sind, sind weniger Speicherzugriffe erforderlich. Das Vergleichen der Gleichungen 9 und 10 mit den Gleichungen 4 und 5 zeigt, daß, wenn alle anderen Dinge gleich sind, die beschnittene vierflächige Interpolation rechentechnisch aufwendiger ist als die radiale Interpolation.

[0113] In **Fig.** 16 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer vierflächigen Interpolation gezeigt. Für dieses Verfahren sind die Eingangsfarbraumwerte **10** aus d Komponenten gebildet. Jede der d Komponenten ist unterteilt in einen Satz von Bits höherer Ordnung **10a** und Bits niedrigerer Ordnung **10b**. Jeder der d Sätze von Bits niedrigerer Ordnung **10b** ist aus n Bits gebildet. Die d Sätze von Bits niedrigerer Ordnung sind jeweils als lb_1, lb_2, lb_3, ... lb_d bezeichnet. Die Bitposition von jedem der Bits niedrigerer Ordnung ist von dem höchstwertigsten Bit zu dem niedrigstwertigsten Bit durch einen Wert von i bezeichnet, der entsprechend von n – 1 bis 0 reicht. Zunächst wird ein Wert gemäß $v[i] = 2^{d-1} \times lb_1[i] + 2^{d-2} \times lb_2[i] + 2^{d-3} \times lb_3[i] + ... + 2^{d-d} \times lb_d[i]$ für i = n – 1 berechnet **800**. Danach wird ein Satz von UND-Werten gemäß v[i] & k, für den Wert von k berechnet **800**, der von 2^d – 1 bis 0 reicht, wobei „&" die bitweise UND-Operation darstellt. Dann wird ein Satz von ODER-Werten gemäß v[i]|k für den Wert von k berechnet **802**, der von 2^d – 1 bis 0 reicht, wobei „|" die bitweise ODER-Operation darstellt. Danach werden 2^d Paare der Scheitelwerte unter Verwendung des Satzes von UND-Werten und des Satzes von ODER-Werten ausgewählt **803**. Jedes der Paare wird unter Verwendung eines UND-Wertes und eines ODER-Wertes ausgewählt, die für einen entsprechenden Wert von k berechnet werden. Schließlich wird ein Satz von 2^d Summen durch Summieren von jedem der 2^d Paare von Scheitelwerten berechnet **804**.

[0114] Das in **Fig.** 16 gezeigte Verfahren ist für eine einzige Iteration der vierflächigen Interpolation. Das Durchführen aufeinanderfolgender Iterationen würde das Berechnen zusätzlicher Werte von v[i], das Berechnen zusätzlicher UND- und ODER-Werte, das Auswählen von Werten von 2^d Summen, die bei der vorherigen Iteration unter Verwendung der zusätzlich berechneten UND- und ODER-Werte berechnet wurden, und das Berechnen zusätzlicher Sätze von 2^d Summen erfordern. Nach der letzen Iteration der vierflächigen Interpolation wird jede der letzten 2^d-Summen durch 2^n geteilt (in **Fig.** 16 nicht gezeigt), wobei n die Anzahl von Iterationen ist und einer der resultierenden Werte als das Ergebnis der Interpolation ausgewählt wird. Die Vision durch 2^n wird nach der letzten Iteration durchgeführt, anstatt des Dividierens durch 2 nach jeder Iteration, um eine Abrundungsfehlerakkumulation zu vermeiden.

[0115] In **Fig.** 17 ist eine Hardwareimplementierung eines vierflächigen Interpolators **900** für die Umwandlung eines Eingangsfarbraumwerts **10** zu einer Komponente von Ausgangsfarbraumwerten **11** gezeigt. Diese gleiche Hardware könnte wiederholt verwendet werden für zusätzliche (D – 1) Durchläufe zum Erzeugen der verbleibenden D – 1 Komponenten des Ausgangsfarbraumwerts **11**. Oder es könnte zusätzliche (D – 1) Duplikationen von einem Teil der Hardwareimplementierung geben, die in **Fig.** 17 gezeigt ist, um jede der D Komponenten gleichzeitig zu erzeugen. Die Hardware, die zum Erzeugen von Multiplexersteuereingangssignalen verwendet wird, könnte für jede der D-Duplikationen verwendet werden. Der in **Fig.** 16 gezeigte vierflächige Interpolator **900** entspricht d = 3 und n = 4 für den Eingangsfarbraumwert **10**. Die in **Fig.** 17 gezeigte Hardwareimplementierung implementiert die Gleichung 8 für die Erzeugung der Unterwürfelscheitelwerte. Der vierflächige Interpolator **900** von **Fig.** 17 ist aus einer ersten, zweiten, dritten und vierten Stufe **901**–**904** gebildet. Jede der vier Stufen **901**–**904** umfaßt 2^3 Addierer, von denen einer als **905** bezeichnet ist. Jede der vier Stufen **901**–**904** umfaßt ferner 2×2^3 Multiplexer, von denen einer als **906** bezeichnet ist, die als 2^3 Paare von Multiplexern angeordnet sind. Schließlich umfaßt jede der vier Stufen **901**–**904** 2^3 bitweise ODER-Blöcke, von denen einer als **907** bezeichnet ist, und 2^3 bitweise UND-Blöcke, von denen einer als **908** bezeichnet ist.

[0116] Jede der Stufen **901**–**904** führt eine Iteration von Interpolation durch. Einige Interpolationsanwendungen können erfordern, daß nur eine einzige Iteration von Interpolation durchgeführt wird. Für eine einzige Iteration der Interpolation ist n = 1. Dies entspricht einer Hardwareimplementierung eines vierflächigen Interpolators **900** unter Verwendung nur der ersten Stufe **901**. Eine zusätzliche Stufe würde für jede zusätzliche Iteration der Interpolation hinzugefügt, die für die spezielle Anwendung erforderlich ist.

[0117] Jedes der Eingangssignale der Multiplexer in der ersten Stufe **901** ist mit den acht Scheitelwerten verbunden, die unter Verwendung der Bits höherer Ordnung **10a** ausgewählt werden. Die zwei Ausgangssignale jedes Paars von Multiplexern in der ersten Stufe **901** sind mit dem ersten und zweiten Eingangssignal des ent-

sprechenden Addierers verbunden. Das Ausgangssignal von jedem der Addierer der ersten Stufe **901** ist der Scheitelwert des ersten Unterwürfels. Wie es vorher erwähnt wurde, wird die Division durch 2 für jede Iteration der Unterwürfelerzeugung verschoben, bis der letzte Unterwürfel erzeugt ist. Die Scheitelwerte des letzten Unterwürfels, der erzeugt wird, werden durch 2^n dividiert, wobei n die Anzahl von Bits in den Bits niedrigerer Ordnung des Eingangsfarbraumwerts ist, und n der Anzahl von Stufen in dem vierflächigen Interpolator entspricht. Die Eingänge jedes Multiplexers für die zweite, dritte und vierte Stufe **902**–**904** sind zu den Ausgängen der Addierer der vorhergehenden Stufe gekoppelt.

[0118] Das Steuereingangssignal von einem der Multiplexer von jedem Paar von Multiplexern ist mit dem Ausgangssignal eines bitweisen ODER-Blocks verbunden. Das Steuereingangssignal des anderen von jedem Paar von Multiplexern ist mit dem Ausgangssignal eines bitweisen UND-Blocks verbunden. Die Multiplexer, die bei dem vierflächigen Interpolator **900** verwendet werden, haben die Fähigkeit, einen von acht 8-Bit-Werten unter Verwendung eines 3-Bit-Steuereingangssignals auszuwählen. Die bitweisen ODER-Blöcke und die bitweisen UND-Blöcke führen jeweils Bit-für-Bit-ODER-Operationen oder UND-Operationen an den Werten durch, die in dieselben eingegeben werden. Für diese d = 3 Implementierung des vierflächigen Interpolators **900** ist jedes der Eingangssignale zu den bitweisen ODER-Blöcken und den bitweisen UND-Blöcken eine 3-Bit-Größe. Das Ausgangssignal von jedem der bitweisen ODER-Blöcke und bitweisen UND-Blöcke an jedem der Multiplexer ist eine 3-Bit-Größe. Die Addierer, die jedem Paar von Multiplexern zugeordnet sind, führen eine Addition der ausgewählten acht-Bit-Werte von jedem der Multiplexer durch.

[0119] Wie es durch die Gleichung 8 angezeigt ist, ist die Scheitelnummer, die dem erzeugten Scheitelwert entspricht, mit einem der Eingangssignale für jeden entsprechenden bitweisen ODER-Block und bitweisen UND-Block verbunden. Weil diese Werte fest sind, können sie zu den richtigen Werten fest verdrahtet werden. Die anderen Eingangssignale für jedes entsprechende Paar von bitweisen UND-Blöcken und bitweisen ODER-Blöcken in einer Stufe sind mit dem Wert von v[i] verbunden, der der Stufe entspricht. Für die erste Stufe **901** ist der Wert v[3]. Für die zweite Stufe **902** ist der Wert v[2] . Für die dritte Stufe **903** ist der Wert v[1]. Für die vierte Stufe **904** ist der Wert v[0].

[0120] Eine Interpolation wird durchgeführt durch Zuführen der Scheitelwerte, die unter Verwendung der Bits höherer Ordnung **10a** ausgewählt werden, zu den Multiplexereingangssignalen der ersten Stufe **900** und Zuführen der geeigneten v[i]-Werte zu den bitweisen ODER-Blöcken und den bitweisen UND-Blöcken jeder Stufe. Der vierflächige Interpolator **900** berechnet die Scheitelwerte für die vier Iterationen der Unterwürfelerzeugung. Die Werte P""(7) bis P""(0) sind die Werte der Scheitel des vierten erzeugten Unterwürfels. Bei diesem Ausführungsbeispiel wird P""(0) ausgewählt, durch 16 dividiert und als das Ergebnis der Interpolation verwendet. Die Teilung durch 16 wird implementiert durch Verschieben von Bits und ist in **Fig.** 17 nicht dargestellt.

[0121] Ein Durchschnittsfachmann auf diesem Gebiet wird erkennen, daß einer der anderen berechneten Werte P""(7) bis P""(1) ausgewählt werden kann, durch 16 dividiert und als das Ergebnis der Interpolation verwendet werden kann. Das Verwenden von Werten, die unterschiedlichen Scheiteln des letzten Unterwürfels entsprechen, um das Interpolationsergebnis zu erzeugen, wird das Ergebnis der Interpolation anders vorspannen. Dies ist eine Überlegung beim Auswählen, welcher Scheitelwert des letzten erzeugten Unterwürfels durch 16 dividiert wird, um das Ergebnis der Interpolation zu erzeugen.

[0122] In **Fig.** 18 ist eine Hardwareimplementierung eines beschnittenen vierflächigen Interpolators **1000** für die Umwandlung der Eingangsfarbraumwerte **10** zu einer Komponente der Ausgangsfarbraumwerte **11** gezeigt. Diese gleiche Hardware könnte wiederholt für alle zusätzlichen (D – 1) Durchläufe verwendet werden, um die verbleibenden D – 1 Komponenten des Ausgangsfarbraumwerts **11** zu erzeugen. Oder es könnte zusätzliche (D – 1) Duplikationen von Teilen der in **Fig.** 18 gezeigten Hardwareimplementierung geben, um jede der D Komponenten gleichzeitig zu erzeugen. Die Hardware, die zum Erzeugen von Multiplexereingangssteuerungen verwendet wird, könnte für jede der D Duplikationen verwendet werden. Der in **Fig.** 18 gezeigte beschnittene vierflächige Interpolator **1000** entspricht d = 3 und n = 4 für den Eingangsfarbraumwert **10**. Der beschnittene vierflächige Interpolator **1000** implementiert die schematische Darstellung der in **Fig.** 15 gezeigten beschnittenen vierflächigen Interpolation. Die Implementierung des beschnittenen vierflächigen Interpolators **1000** erfordert wesentlich weniger Hardware als die Implementierung des vierflächigen Interpolators **900**.

[0123] Für d = 3 führen die bitweisen UND-Blöcke **1001a** bis **1001k** jeweils eine Bit-für-Bit-UND-Operation an den 3-Bit-Eingangsgrößen durch, um 3-Bit-Ausgangssignalgrößen zu erzeugen. Gleichartig dazu führen für d = 3 bitweise ODER-Blöcke **1002a** bis **1002k** jeweils eine Bit-für-Bit-ODER-Operation an den 3-Bit-Eingangssignalgrößen durch, um 3-Bit-Ausgangssignalgrößen zu erzeugen. Jedes der 3-Bit-Ausgangssignale der bitweisen UND-Blöcke **1001a** bis **1001g** und der bitweisen ODER-Blöcke **1002a**–**1002g** wird verwendet, um die Aus-

wahl von einer von acht 8-Bit-Größen in dem entsprechenden der Multiplexer **1003a** bis **1003n** auszuwählen. Die Ausgangssignale von jedem der Multiplexer **1003a**–**1003o** sind mit den Eingangssignalen der Addierer **1004a**–**1004h** verbunden.

[0124] Eine Interpolationsoperation wird unter Verwendung des beschnittenen vierflächigen Interpolators **1000** durchgeführt, durch Zuführen der Scheitelwerte, die unter Verwendung der Bits höherer Ordnung **10a** ausgewählt wurden, zu den Eingangssignalen der Multiplexer **1003a**–**1003o** und Zuführen der berechneten Werte von v[i] zu den Eingangssignalen der bitweisen ODER-Blöcke **1002a**–**1002k** und der bitweisen UND-Blöcke **1001a**–**1001k**, wie es in **Fig.** 18 gezeigt ist. Außerdem wird ein Scheitelwert, der unter Verwendung der Bits höherer Ordnung **10a** ausgewählt wird, zu dem Eingangssignal des Addierers **1004h** zugeführt. Unter Verwendung der berechneten Werte von v[i] berechnen die bitweisen UND-Blöcke **1001a**–**1001k** und die bitweisen ODER-Blöcke **1002a**–**1002k** die Werte, die in die Steuereingangssignale der Multiplexer **1003a**–**1003n** eingegeben werden. Der Multiplexer **1003o** verwendet v[3] direkt. Die Werte, die durch die Multiplexer **1003a**–**1003o** ausgewählt werden, sind diejenigen, die notwendig sind, um das Interpolationsergebnis gemäß der schematischen Darstellung der in **Fig.** 15 gezeigten beschnittenen vierflächigen Interpolation zu berechnen. Die Scheitelwerte, die durch die Multiplexer **1003a**–**1003o** ausgewählt werden, werden für eine Summierung an die Eingänge der Addierer **1004a**–**1004h** gesendet. Das Ausgangssignal des letzten Addierers in der Kette von Additionen wird durch 16 dividiert und als das Ergebnis der Interpolation verwendet. Die Division durch 16 wird durch Bitverschiebung erreicht und ist in **Fig.** 18 nicht gezeigt.

[0125] Der in **Fig.** 18 gezeigte beschnittene vierflächige Interpolation ist für d = 3 und n = 4 implementiert. Für einige Anwendungen können weniger als vier Iterationen der Interpolation ausreichend sein. Andere Anwendungen können mehr als vier Iterationen der Interpolation erfordern. Die Hardwareimplementierung der beschnittenen vierflächigen Interpolation für d = 3 und n = 1 würde nur einen einzigen Addierer und einen einzigen Multiplexer verwenden, um P'(0) zu erzeugen, wie es in **Fig.** 18 gezeigt ist. Die Hardwareimplementierungen für n = 2 und n = 3 erfordern zum Erzeugen von P''(0) bzw. P'''(0), wie es in **Fig.** 18 gezeigt ist, mehr bitweise UND-Blöcke, bitweise ODER-Blöcke, Multiplexer und Addierer.

[0126] In **Fig.** 19 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer beschnittenen vierflächigen Interpolation gezeigt. Für dieses Verfahren sind die Eingangsfarbraumwerte **10** aus d Komponenten gebildet. Jede der d Komponenten ist in einen Satz von Bits höherer Ordnung **10a** und Bits niedrigerer Ordnung **10b** unterteilt. Jeder der d Sätze von Bits niedrigerer Ordnung **10b** ist aus n Bits gebildet. Die d Sätze von Bits niedrigerer Ordnung sind jeweils als lb_1, lb_2, lb3, ... lb_d bezeichnet. Die Bitposition von jedem der Bits niedrigerer Ordnung wird von dem höchstwertigsten Bit zu dem niedrigstwertigsten Bit durch einen Wert von i bestimmt, der entsprechend von n – 1 bis 0 reicht. Zunächst werden $2^n – 2$ Werte gemäß den Typen von bitweisen UND- und bitweisen ODER-Operationen, die in **Fig.** 15 gezeigt sind, mit $v[i] = 2^{d–1} \times lb_l[i] + 2^{d–2} \times lb_2[i] + 2^{d–3} \times lb_3[i] + ... + 2^{d–d} \times lb_d[i]$ für i berechnet **1100**, das von n – 1 bis 0 reicht. Nachfolgend wird das Minimum von 2^n und 2^d von Interpolationsdatenwerten unter Verwendung eindeutiger $2^n – 2$ Werte, die im Schritt **1100** berechnet werden, einem Wert von v[n – 1] und Bits höherer Ordnung **10a** ausgewählt **1101**. Dann wird die Summe der Interpolationsdatenwerte, die im Schritt **1101** ausgewählt wurden, berechnet **1102**.

[0127] Das in **Fig.** 19 gezeigte Verfahren ist für n Iterationen der beschnittenen vierflächigen Interpolation. Die Summe, die im Schritt **1102** berechnet wird, wird durch 2^n dividiert, um das Ergebnis zu erzeugen. Diese Division ist in **Fig.** 19 nicht gezeigt.

[0128] In **Fig.** 20 ist ein Flußdiagramm hoher Ebene eines Verfahrens gezeigt, das in Software implementiert ist, um eine beschnittene vierflächige Interpolation durchzuführen. Zunächst wird eine Bestimmung **1200** durchgeführt, ob eine der Komponenten des Eingangsfarbraumwerts (a, b, c) **10** einer Position auf einer äußeren Grenze des Würfelgitters 1 entspricht. Dies ist der Fall, falls eine oder mehrere der Komponenten des Eingangsfarbraumwerts einen hexadezimalen Wert von FF aufweist. Falls dies der Fall ist, dann wird zum Zweck des Erzeugens des Index in dem Würfelgitter 1 zum Wiedergewinnen der notwendigen Scheitelwerte, den Komponenten des Eingangsfarbraumwerts **10**, die einen hexadezimalen Wert von FF aufweisen, ein hexadezimaler Wert von **100** zugewiesen **1201**.

[0129] Nachfolgend wird auf die Versätze von dem Ursprung des Würfels in dem Würfelgitter 1 zugegriffen, unter Verwendung Bits höherer Ordnung **10a**, und die Unterwürfel, die, während der beschnittenen vierflächigen Interpolation erzeugt werden, werden unter Verwendung der Beziehungen berechnet **1202**, die in der schematischen Darstellung der beschnittenen vierflächigen Interpolation von **Fig.** 15 gezeigt sind. Dann werden die Indizes, die zum Zugreifen auf die Werte verwendet werden, die den erforderlichen Scheiteln des ausgewählten Würfels in dem Würfelgitter in einer Nachschlagtabelle entsprechen, berechnet **1203**. Schließlich

werden die Werte für jede Komponente des Ausgangsfarbraumwerts (x, y, z) **11** berechnet **1204**. In **Fig.** 40 dieser Beschreibung ist der Code einer Implementierung bei C für n = 4 des Verfahrens hoher Ebene der in **Fig.** 20 gezeigten beschnittenen vierflächigen Interpolation vorgesehen.

[0130] Es ist auch möglich, eine beschnittene vierflächige Interpolation in Hardware zu implementieren. Die rechentechnischen Effizienzen, die in der Softwareimplementierung einer beschnittenen vierflächigen Interpolation existierten, liegen in der Hardwareimplementierung als reduzierte Hardwareanforderungen vor. Wie es vorher erwähnt wurde, werden Verschiebungen und Verkettungen implementiert, ohne daß zusätzliche Hardwareelemente erforderlich sind. Die Hardwarefunktionsblöcke, die erforderlich sind, um die beschnittene vierflächige Interpolation durchzuführen, umfassen Addierer, UND-Gatter, ODER-Gatter und Multiplexer. Mit D Abmessungen in dem Ausgangsfarbraum, d Abmessungen in dem Eingangsfarbraum und n Bits, die jede Gruppe von Bits niedrigerer Ordnung **10b** des Eingangsfarbraumwerts darstellen, können die Anforderungen der Hardwareimplementierung der beschnittenen Unterteilungsinterpolation berechnet werden als:

Gl. 11 $$\text{Anzahl von Addierern} = D \times 2^n$$

Gl. 12 $$\text{Anzahl von Multiplexern} = D \times (2^n - 1)$$

Gl. 13 $$\text{Anzahl von zusätzlichen UND/ODER-Gattern} = d \times \left\{ 2 \times \left[\sum_{i=1}^{n-1} \left(2^i - 1 \right) \right] \right\}$$

[0131] Um den Gatterpegelentwurf zu erzeugen, der notwendig ist, um die beschnittene radiale Interpolation in Hardware zu implementieren, kann eine üblicherweise verwendete Hardwarebeschreibungssprache, wie z. B. VHDL, verwendet werden. In **Fig.** 41 ist eine Liste des VHDL-Codes enthalten, der eine Hardwareimplementierung einer beschnittenen radialen Interpolation erzeugen kann.

[0132] Wie es vorher erörtert wurde, kann die radiale Interpolation bei Umwandlungen zwischen dem RGB- und dem CMY-Farbraum für bestimmte Eingangsfarbraumwerte zu Druckartefakten führen. Aufgrund dieser Druckartefakte kann es sein, daß eine vierflächige Interpolation wünschenswertere Ergebnisse erzielt. Um die Komplexität zu reduzieren, kann die vierflächige Interpolation unter Verwendung einer beschnittenen vierflächigen Interpolation implementiert werden, obwohl diese Interpolationstechnik nach wie vor rechentechnisch intensiver ist als die radiale Interpolation.

[0133] Für Umwandlungen zwischen anderen Farbräumen (wie z. B. CieLab, LUV oder YC_bC_r) kann die radiale Interpolation vorzuziehen sein, weil dieselbe adäquate Ergebnisse erzielt und rechentechnisch sehr effizient ist. Außerdem ist es möglich, daß eine radiale Interpolation in einigen Fällen tatsächlich angenehmere Ergebnisse erzielt als die vierflächige Interpolation. Falls die Interpolationsverfahren in Software implementiert sind, wird das Verwenden alternativer Verfahren ohne weiteres durch Aufrufen unterschiedlicher Routinen durchgeführt. Das Implementieren verschiedener Interpolationsverfahren in Hardware kann jedoch eine getrennte Logik erfordern. Weil die getrennten Hardwareimplementierungen der beiden Interpolationstechniken nicht voll ausgenutzt werden, ist diese Lösung aufwendig.

[0134] Eine übliche Hardwareimplementierung liefert die Fähigkeit zum alternativen Durchführen einer radialen Interpolation und einer beschnittenen vierflächigen Interpolation mit weniger Hardware als bei einer getrennten Hardwareimplementierung dieser Interpolationstechniken. In **Fig.** 21 ist eine schematische Darstellung einer üblichen radialen Interpolation und einer beschnittenen vierflächigen Interpolationsimplementierung gezeigt. Wie es in **Fig.** 21 angezeigt ist, wird die durchgeführte Interpolationstechnik durch die Scheitelwerte bestimmt, die in die Hardware eingegeben werden. Normalerweise ist die Anzahl von Bits, die verwendet werden, um jeden v(i) Term auszudrücken, kleiner als die Anzahl von Bits, die verwendet werden, um jeden P[v(i)] Term auszudrücken. Deswegen ist es im allgemeinen weniger komplex, die v(i) Terme vor dem Speicherzugriff zu multiplexen, um die Werte wiederzugewinnen, die den P[v(i)] Termen zugeordnet sind.

[0135] Von **Fig.** 21 sollte klar sein, daß zwei der Scheitelwerte, die sowohl für die radiale Interpolation als auch die beschnittene vierflächige Interpolation verwendet werden, für alle Werte von n gleich sind. Daher erfordert eine gemeinsame Hardwareimplementierung der radialen Interpolationen der beschnittenen vierflächigen Interpolation, daß zusätzliche $2^n - 2$ Multiplexer (die jeweils d Steuerbits aufweisen) zu der Hardwareimplementierung der beschnittenen vierflächigen Interpolation addiert werden. In **Fig.** 42 ist der VHDL-Code für eine Hardwareimplementierung der gemeinsamen radialen Interpolation und beschnittenen vierflächigen Inter-

polation enthalten.

[0136] In **Fig.** 22 ist eine Hardwareimplementierung eines gemeinsamen beschnittenen radialen und beschnittenen vierflächigen Interpolators **1300** gezeigt. Die Hardwareimplementierung einer gemeinsamen beschnittenen radialen und beschnittenen vierflächigen Interpolation ist ähnlich wie diejenige der beschnittenen vierflächigen Interpolation. Der Unterschied ist die Addition von 14 Multiplexern **1301a**–**1301n**, die verwendet wird, um die Daten zu den Steuereingängen der Multiplexer **1302a**–**1302n** auszuwählen. Ein einzelnes Bit wird verwendet, um die Auswahl der Daten an den Eingängen der Multiplexer **1301a**–**1301n** zu steuern. Das einzelne Bit steuert, ob die Multiplexereingangssignale in die Multiplexer **1302a**–**1302n** für eine beschnittene vierflächige Interpolation oder für eine beschnittene radiale Interpolation sind. Das Multiplexersteuereingangssignal für die Multiplexer **1302a**–**1302n** bestimmt, welche der Interpolationsdatenwerte zu den Addierern gekoppelt werden. Wenn das Bit in dem ersten seiner beiden Zustände ist, arbeitet die Hardware von **Fig.** 22 als ein beschnittener radialer Interpolator. Wenn das Bit in dem zweiten seiner beiden Zustände ist, arbeitet die Hardware von **Fig.** 22 als ein beschnittener vierflächiger Interpolator.

[0137] Das Steuerbit für die Multiplexer **1301a**–**1301n** wird verwendet, um zwischen den Werten von v(i) und den Werten auszuwählen, die unter Verwendung der bitweisen ODER-Blöcke **1303a**–**1303k** und den bitweisen UND-Blöcken **1304a**–**1304k** berechnet wurden. Die Addierer **1305a**–**1305o** summieren die Ausgangssignale der Multiplexer **1302a**–**1302o**. Durch Verschieben von Bits wird die resultierende Summe durch 16 dividiert (in **Fig.** 22 nicht gezeigt), um das Ergebnis zu erzeugen.

[0138] Die Hardware von **Fig.** 22 könnte wiederholt für zusätzliche (D – 1) Durchläufe verwendet werden, um die verbleibenden D – 1 Komponenten des Ausgangsfarbraumwerts **11** zu erzeugen. Oder es könnte zusätzliche (D – 1) Duplikationen von Teilen der in **Fig.** 22 gezeigten Hardwareimplementierung geben, um jede der D Komponenten gleichzeitig zu erzeugen. Die Hardware, die zum Erzeugen von Multiplexersteuereingangssignalen verwendet wird, könnte für jede der D-Duplikationen verwendet werden. Der gemeinsame beschnittene radiale und beschnittene vierflächige Interpolator **1300**, der in **Fig.** 22 gezeigt ist, entspricht d = 3 und n = 4 für den Eingangsfarbraumwert **10**. Der gemeinsame beschnittene radiale und beschnittene vierflächige Interpolator **1300** implementiert die schematische Darstellung der in **Fig.** 21 gezeigten gemeinsamen beschnittenen radialen Lind beschnittenen vierflächigen Interpolation.

[0139] Bei dem Interpolationsprozeß bilden die Bits höherer Ordnung **10a** eines Eingangsfarbraumwerts **10** einen Index, der verwendet wird, um auf Interpolationsdatenwerte zuzugreifen. Die Interpolation wird unter Verwendung der Bits niedrigerer Ordnung **10b** des Eingangsfarbraumwerts durchgeführt. Wie es vorher erwähnt wurde, entsprechen die zugegriffenen Werte dem Scheitel eines Würfels in einem Würfelgitter 1. Abhängig von den Charaktiristika des Ausgangsfarbraums können die Werte, die den Scheiteln des zugegriffenen Würfels zugeordnet sind, bei Raten variieren, die von der Abmessung des Würfelgitters 1 oder von der Region des Würfelgitters 1 abhängen, in der der ausgewählte Würfel positioniert ist. Aufgrund dieser Möglichkeit können verbesserte Interpolationsergebnisse erzeugt werden, durch Variieren der Interpolationsauflösung zwischen Werten der Scheitel in dem gesamten Würfelgitter 1, die den variierenden Änderungsraten zwischen den Scheitelwerten entsprechen. Das Einstellen der Interpolationsauflösung auf der Basis der Position des ausgewählten Würfels in dem Würfelgitter 1 kann implementiert werden, indem es ermöglicht wird, daß die Unterteilung des Eingangsfarbraumwerts **10** in Bits höherer Ordnung **10a** und Bits niedrigerer Ordnung **10b** variieren kann. In Regionen des Würfelgitters 1 mit hohen nichtlinearen Änderungsraten bei den Scheitelwerten wären die Unterschiede zwischen den Scheitelwerten relativ groß, unter Verwendung einer Anzahl von Bits, die nicht an die Farbraumcharakteristika angepaßt sind, um jede Komponente der Bits höherer Ordnung **10a** darzustellen. Um die Werte zwischen den Scheiteln zu reduzieren, wird eine größere Anzahl von Bits verwendet, um die Komponenten der Bits höherer Ordnung **10a** darzustellen. Folglich wird in diesen Regionen eine kleinere Anzahl von Bits verwendet, um die Bits niedrigerer Ordnung **10b** darzustellen. Für Regionen des Würfelgitters 1 mit geringeren Änderungsraten oder linearen Änderungsraten bei den Scheitelwerten wären die Unterschiede zwischen den Scheitelwerten unter Verwendung einer Anzahl von Bits, die nicht an die Farbraumcharakteristika angepaßt sind, um jede Komponente der Bits höherer Ordnung **10a** darzustellen, relativ klein oder relativ linear. Um die Werte zwischen den Scheiteln zu erhöhen, wird eine kleinere Anzahl von Bits verwendet, um die Komponenten der Bits höherer Ordnung **10a** darzustellen.

[0140] Um eine Interpolationstechnik zu implementieren, die eine variierende Interpolationsauflösung über dem Ausgangsfarbraum unterstützen kann, der durch das Würfelgitter 1 dargestellt ist, muß die Interpolationstechnik die wechselnde Anzahl von Bits unterbringen, die verwendet werden, um die Komponenten der Bits niedrigerer Ordnung **10b** darzustellen. Um dies zu erreichen, ist ein Wert (n, p, q) definiert, so daß n Bits verwendet werden, um zwischen Gitterpunkten in der „a"-Abmessung zu interpolieren, p Bits werden verwendet,

um zwischen Gitterpunkten in der b-Abmessung zu interpolieren, und q Bits werden verwendet, um zwischen Gitterpunkten in der c-Abmessung zu interpolieren. Es ist möglich, die Werte sowohl von n, p als auch q zu beschränken, so daß dieselben über die Gesamtheit des Würfelgitters 1 festgelegt sind, oder es zu ermöglichen, daß jeder der Werte von n, p und q unabhängig oder abhängig voneinander durch die Regionen des Würfelgitters 1 variieren.

[0141] In **Fig.** 23a bis 23e ist eine graphische Darstellung eines nichtsymmetrischen radialen Interpolationsprozesses gezeigt, der vier Bits zum Darstellen der a_l Komponente, drei Bits zum Darstellen der b_l Komponente und zwei Bits zum Darstellen der c_l Komponente der Bits niedrigerer Ordnung **10b** des Eingangsfarbraumwerts verwendet. Die erste Iteration der Würfelunterteilung ist die Auswahl eines Unterwürfels, der eine Hälfte des Würfels besetzt, der durch die Bits höherer Ordnung **10a** ausgewählt wird, unter Verwendung des Bits von a_l, das der i = 3 Position entspricht. Die zweite Iteration der Würfelunterteilung ist die Auswahl eines Unterwürfels, der ein Viertel des vorhergehenden Unterwürfels besetzt, unter Verwendung jeweils eines Bits von a_l und b_l. Die dritte Iteration der Würfelunterteilung ist die Auswahl eines Unterwürfels, der ein Achtel des vorhergehenden Unterwürfels besetzt, unter Verwendung jeweils eines Bits von a_l, b_l und c_l. Die vierte Iteration der Würfelunterteilung ist ebenfalls die Auswahl eines Unterwürfels, der ein Achtel des vorhergehenden Unterwürfels besetzt, unter Verwendung jeweils eines Bits von a_l, b_l und c_l. Wie es daraus ersichtlich ist, bestimmt die Anzahl von Bits der Komponenten der Bits niedrigerer Ordnung **10b**, die verfügbar sind, um den Unterwürfel zu erzeugen, den Bruchteil des Würfels, der verwendet wird, um den Unterwürfel zu erzeugen, der durch den Unterwürfel besetzt wird.

[0142] In Tabelle 4 und in den Gleichungen 14 bis 19 sind die Beziehungen gezeigt, die notwendig sind, um die Werte der Unterwürfelscheitel für jede Iteration der radialen Unterwürfelerzeugung zu berechnen. Die Gleichungen 14 bis 19 erzeugen die Werte, die bei den in Tabelle 4 gezeigten Beziehungen verwendet werden, so daß die korrekten Unterwürfelscheitelwerte erzeugt werden mit oder ohne den entsprechenden Bits von a_l, b_l und c_l, die für diese Iteration der Unterwürfelerzeugung vorliegen. Falls für eine gegebene Iteration einer Unterwürfelerzeugung ein Bit in einem oder mehreren von a_l, b_l oder c_l nicht vorliegt, erzeugt die Gleichung 19 die Nummer des Scheitels des Würfels, der beim Erzeugen eines Scheitelwerts des Unterwürfels verwendet wird, um das/die fehlende(n) Bit (s) auszugleichen.

Gl. 14 $$Maske_a = (2^n - 1)$$

Gl. 15 $$Maske_b = (2^p - 1)$$

Gl. 16 $$Maske_c = (2^q - 1)$$

Gl. 17 $$m[i] = (4 \times Maske_a[i]) + (2\ Maske_b[i]) + Maske_c[i]$$

Gl. 18 $$v[i] = m[i]\ \&\ \{(4 \times a[i]) + (2 \times b[i]) + c[i]\}$$

Gl. 19 $$f(N, i) = v[i]\ |\ (N\ \&\sim m[i])$$

Tabelle 4

P'[7] = {P[7] + P[f(7, i)]} ÷ 2
P'[6] = {P[6] + P[f(6, i)]} ÷ 2
P'[5] = {P[5] + P[f(5, i)]} ÷ 2
P'[4] = {P[4] + P[f(4, i)]} ÷ 2
P'[3] = {P[3] + P[f(3, i)]} ÷ 2
P'[2] = {P[2] + P[f(2, i)]} ÷ 2
P'[1] = {P[1] + P[f(1, i)]} ÷ 2
P'[0] = {P[0] + P[f(0, i)]} ÷ 2

[0143] In **Fig.** 24 ist eine graphische Darstellung der Erzeugung eines Unterwürfels **1400** von einem Würfel **1401** gezeigt, unter Verwendung einer nichtsymmetrischen radialen Unterwürfelerzeugung. Bei der speziellen Iteration der in **Fig.** 15 gezeigten Unterwürfelerzeugung liegt das Bit, das der Iteration für die Komponenten b_l und c_l entspricht, nicht vor. Daher geben die Beziehungen in Tabelle 4 mit f(N, i), berechnet für die Bits von b_l und c_l, die der nichtvorliegenden Iteration entsprechen, vor, daß die Scheitelwerte P'[7], P'[6], P'[5] und P'[4] als der Mittelwert der Werte der beiden Scheitel berechnet wird, die vertikal jeweils mit P'[7], P'[6], P'[5] und P'[4] ausgerichtet sind. Die Berechnung der Scheitelwerte für andere Kombinationen, bei denen jeweils a_l, b_l

oder c_l vorliegen oder nicht, werden analog durch die Gleichung 19 gehandhabt.

[0144] In **Fig.** 25 ist eine schematische Darstellung der nichtsymmetrischen beschnittenen radialen Interpolationsberechnung gezeigt. Die Anzahl von Berechnungen, die erforderlich sind, um die nichtsymmetrische beschnittene radiale Interpolation durchzuführen, wird berechnet als:

Gl. 20

$$\text{Anzahl von ALU - Operationen} = \{n \times [(4 \times d) - 1]\} + \left\{3 \times \left[2^{n-1} \times \sum_{i=1}^{n-1} \left(1 - 2^{-i}\right)\right] + [D \times (2^n + 1)]\right\}$$

[0145] In **Fig.** 25 sind 2^n Speicherzugriffe gezeigt. Ein ausgewählter Würfel hat jedoch ein Maximum von 2^d Scheitelwerten, wobei d die Anzahl von Abmessungen des Eingangsfarbraumwerts **10** ist. Daher sind einige der 2^n (16 Werte für n = 4) Werte, die an den Eingängen des Diagramms von **Fig.** 25 gezeigt sind, redundant. Es folgt, daß für 2^n größer als 2^d die Anzahl von Speicherzugriffen, die durchgeführt wird, auf die Anzahl von Scheitel in dem Würfel für jede Abmessung des Ausgangssignals des Farbraums beschränkt werden kann. Daher ist die Anzahl von Speicherzugriffen, die für D Abmessungen in dem Ausgangsfarbraum erforderlich ist:

Gl. 21

$$\text{Anzahl von Speicherzugriffen} = D \times \min(2^d, 2^n$$

)

[0146] In **Fig.** 26 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer nichtsymmetrischen beschnittenen radialen Interpolation gezeigt. Für dieses Verfahren sind die Eingangsfarbraumwerte **10** aus d Komponenten gebildet. Jede der d Komponenten ist in einen Satz von Bits höherer Ordnung **10a** und Bits niedrigerer Ordnung **10b** unterteilt. Die d Sätze von Bits niedrigerer Ordnung sind jeweils als lb_1, lb_2, lb_3, ... lb_d bezeichnet. Jeder der d Sätze von Bits niedrigerer Ordnung **10b** ist jeweils aus n_1, n_2, n_3, ... n_d Bits gebildet. Die Bitposition von jedem der d Sätze von Bits niedrigerer Ordnung wird von dem höchstwertigsten Bit zu dem niedrigstwertigsten Bit durch entsprechende Werte von i_1, i_2, i_3, ... i_d bestimmt, die jeweils entsprechend von $n_1 - 1$ bis 0, $n_2 - 1$ bis 0, $n_3 - 1$ bis 0, ... $n_d - 1$ bis 0 reichen. Zunächst wird ein Satz von $2^n - n - 1$ Werten unter Verwendung von f(N, i) = v[i]| (N &~ m[i]) berechnet **1500**, wobei $m[i] = 2^{d-1} \times Maske_1[i] + 2^{d-2} \times Maske_2[i] + 2^{d-3} \times Maske_3[i] + ... + 2^{d-d} \times Maske_d[i]$. Für diese Berechnung sind die Werte von $Maske_j = 2^k - 1$ jeweils für einen Wert k berechnet, der aus n_1, n_2, n_3, ... n_d berechnet wird, wobei der Wert von j dem Wert des tiefergestellten Zeichens des ausgewählten von n_1, n_2, n_3, ... n_d entspricht. Die Werte von j reichen von 1 bis d. Die Werte von v[i] werden als $m[i] \& (2^{d-1} \times lb_1[i] + 2^{d-2} \times lb_2[i] + 2^{d-3} \times lb_3[i] + ... + 2^{d-d} \times lb_d[i])$ für Werte von i berechnet, die von n – 1 bis 0 reichen, wobei n gleich dem höchsten von n_1, n_2, n_3, ... n_d ist. Der Wert von N entspricht den Scheitelnummern, die von 1 bis 2^d reichen. Nachfolgend wird eine Anzahl von Interpolationsdatenwerten gleich dem Minimum von 2^n und 2^d ausgewählt **1501** unter Verwendung der eindeutigen Werte in dem Satz von $2^n - n - 1$ Werten, der Werte von v[i] für i, die von n – 1 bis 0 reichen, und den d Sätzen von Bits höherer Ordnung. Schließlich werden die ausgewählten Interpolationsdatenwerte addiert **1502**, um eine Summe zu erzeugen. Um einen Abrundungsfehler zu erzeugen, wird die erzeugte Summe durch 2^n dividiert. Dieser Schritt ist in **Fig.** 26 nicht gezeigt.

[0147] In **Fig.** 27 ist ein Flußdiagramm hoher Ebene eines Verfahrens gezeigt, das in Software implementiert ist, um eine nichtsymmetrische radiale Interpolation durchzuführen. Zunächst werden die Maskenwerte für jede Komponente des Eingangsfarbraumwerts **10** erzeugt **1600**. Danach wird eine Bestimmung **1601** durchgeführt, ob irgendeine der Komponenten des Eingangsfarbraumwerts (a, b, c) **10** einer Position auf einer äußeren Grenze des Würfelgitters 1 entspricht. Dies ist der Fall, falls eine oder mehrere der Komponenten des Eingangsfarbraumwerts einen hexadezimalen Wert von FF aufweist. Falls dies der Fall ist, dann wird zum Zweck des Erzeugens der Indizes zum Wiedergewinnen der notwendigen Scheitelwerte den Komponenten des Eingangsfarbraumwerts **10**, die einen hexadezimalen Wert von FF aufweisen, ein hexadezimaler Wert von **100** zugewiesen **1602**. Dann werden die Werte sowohl von m[i] als auch v[i] berechnet **1603**. Danach werden die Indizes, die verwendet werden, um auf jeden der Scheitelwerte zuzugreifen, die für die Interpolation verwendet werden, berechnet **1604**. Schließlich wird jede der Komponenten des Ausgangsfarbraumwerts unter Verwendung der Werte berechnet **1605**, auf die durch die Indizes zugegriffen wird, die im Schritt **1604** berechnet werden.

[0148] In **Fig.** 43 ist eine Auflistung in C des Codes für eine Softwareimplementierung einer nichtsymmetrischen radialen Interpolation enthalten. Für die nichtsymmetrische radiale Interpolation entsprechen die berechneten Indizes Versätzen von dem Ursprung des Würfels, der durch die Bits höherer Ordnung **10a** ausge-

wählt wird. Aufgrund der sich ändernden Auflösung, die in dem Ausgangsfarbraum verwendet wird, werden die Werte der Scheitel für die drei Würfel (ein Würfel für jede Abmessung des Ausgangsfarbraums), die durch die Bits höherer Ordnung **10a** ausgewählt werden, in die Routine von **Fig.** 43 geleitet, jedesmal wenn eine Farbraumumwandlung einem Eingangsfarbraumwert **10** durchgeführt wird. Dies unterscheidet sich von dem Code für die beschnittene radiale und beschnittene vierflächige Interpolation, bei der die Farbtabelle als ein Array in die Routine geleitet wird, und Indizes in diese Tabelle in der Routine berechnet werden.

[0149] Es ist auch möglich, eine nichtsymmetrische radiale Interpolation in Hardware zu implementieren. Wie es vorher erwähnt wurde, werden Verschiebungen und Verkettungen implementiert, ohne zusätzliche Hardwareelemente zu erfordern. Die Hardwarefunktionsblöcke, die erforderlich sind, um die nichtsymmetrische radiale Interpolation durchzuführen, umfassen Addierer, UND-Gatter, ODER-Gatter und Multiplexer. Mit D Abmessungen in dem Ausgangsfarbraum, d Abmessungen in dem Eingangsfarbraum und n Bits, die die maximale Anzahl von Bits darstellen, die verwendet werden, um eine der Komponenten des Eingangsfarbraumwerts darzustellen, können die Anforderungen der Hardwareimplementierung der nichtsymmetrischen radialen Interpolation berechnet werden als:

Gl. 22 $$\text{Anzahl von Addierern} = D \times [(2^n - 1) + 1]$$

Gl. 23 $$\text{Anzahl von Multiplexern} = D \times (2^n - 1)$$

Gl. 24 $$\text{Anzahl von zusätzlichen UND/ODER - Gattern} = (3 \times d) \times \left\{(2^n - 1) \times \sum_{i=1}^{n-1} \left(1 - 2^{-i} - 1\right)\right\}$$

[0150] Um den Gatterpegelentwurf zu erzeugen, der notwendig ist, um die nichtsymmetrische beschnittene radiale Interpolation in Hardware zu implementieren, kann eine allgemein verwendete Hardwarebeschreibungssprache, wie z. B. VHDL, verwendet werden. In **Fig.** 44 ist eine Auflistung des VHDL-Codes enthalten, der eine Hardwareimplementierung einer nichtsymmetrischen beschnittenen radialen Interpolation erzeugen kann.

[0151] In **Fig.** 28 ist eine Hardwareimplementierung eines nichtsymmetrischen beschnittenen radialen Interpolators **1700** gezeigt. Steuereingangssignal-Berechnungsblöcke **1701a**–**1701k** berechnen die Werte, die durch die Steuereingänge der Multiplexer **1702a**–**1702o** verwendet werden, die zu den Steuereingangssignal-Berechnungsblöcken **1701a**–**1701k** gekoppelt sind. Jeder der Steuereingngssignal-Berechnungsblöcke **1701a**–**1701k** führt die Berechnung der Gleichungen 14 bis 19 an dem Eingangssignal zu diesem Steuereingangssignal-Berechnungsblock durch. Wie es in **Fig.** 28 gezeigt ist, verwenden einige der Steuereingangsignale der Multiplexer **1702a**–**1702o** Werte von v[i]. Die Addierer **1703a**–**1703o** summieren die Ausgangssignale der Multiplexer **1702a**–**1702o**. Diese Summe wird durch Bitverschiebung (in **Fig.** 28 nicht gezeigt) durch 2^n dividiert, um das Interpolationsergebnis zu erzeugen.

[0152] Die Hardware von **Fig.** 28 könnte wiederholt für zusätzliche (D – 1) Durchläufe verwendet werden, um die verbleibenden D – 1 Komponenten des Ausgangsfarbraumwerts **11** zu erzeugen. Oder es könnte zusätzliche (D – 1) Duplikationen von Teilen der in **Fig.** 28 gezeigten Hardwareimplementierung geben, um jede der D Komponenten gleichzeitig zu erzeugen. Die Hardware, die zum Erzeugen von Multiplexersteuereingangssignalen verwendet wird, könnte für jede der D Duplikationen verwendet werden. Der in **Fig.** 28 gezeigte nichtsymmetrische beschnittene radiale Interpolator **1700** entspricht d = 3 und dem Maximum von einem von n_1, n_2, n_3, ... n_d gleich 4 für den Eingangsfarbraumwert **10**. Der nichtsymmetrische beschnittene radiale Interpolator **1700** implementiert die schematische Darstellung der in **Fig.** 25 gezeigten beschnittenen vierflächigen Interpolation.

[0153] In **Fig.** 29 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer nichtsymmetrischen radialen Interpolation unter Verwendung der Gleichungen von Tabelle 4 gezeigt. Zunächst wird ein erster Satz von 2^d Werten unter Verwendung von f(N, i) = v[i]|(N &~ m[i]) berechnet **1800**. Nachfolgend werden 2^d Paare von Interpolationsdatenwerten ausgewählt **1801**, wobei jedes Paar von dem Interpolationsdatenwert gebildet ist, der unter Verwendung von einem der Sätze von 2^d Werten und dem Interpolationsdatenwert ausgewählt wurde, der dem Scheitel entspricht, der durch den Wert von N bestimmt ist. Schließlich werden 2^d Summen von den ausgewählten 2^d Paaren von Interpolationsdatenwerten berechnet **1802**.

[0154] Das in **Fig.** 29 gezeigte Verfahren ist für eine einzelne Iteration einer nichtsymmetrischen radialen In-

terpolation. Es sollte klar sein, daß weitere Iterationen durchgeführt würden durch Wiederholen der Schritte von **Fig.** 29 mit den aufeinanderfolgenden Sätzen von 2^d Werten, die unter Verwendung von f(N, i) für Werte von v[i] und m[i] berechnet werden, die aufeinanderfolgend dekrementierten Werten von i entsprechen, Auswählen aufeinanderfolgender Sätze von 2^d Paaren von Werten von dem vorher berechneten Satz von 2^d Summen und Berechnen aufeinanderfolgender Sätze von 2^d Summen von den aufeinanderfolgenden Sätzen von 2^d Paaren von Werten. Nach dem Durchführen von n Iterationen, wo n gleich dem höchsten von n_1, n_2, n_3, ... n_d ist, wird eine der 2^d Summen des letzten Satzes, der berechnet wurde, durch 2^n dividiert, um das Ergebnis zu erzeugen. Die Division durch 2, die nach jeder Iteration durchgeführt werden könnte, wird bis nach der letzten Iteration verzögert, um einen Abrundungsfehler zu vermeiden.

[0155] In **Fig.** 30 ist eine Hardwareimplementierung eines nichtsymmetrischen radialen Interpolators **1900** für die Umwandlung von Eingangsfarbraumwerten **10** zu einer Komponente von Ausgangsfarbraumwerten **11** gezeigt. Diese gleiche Hardware könnte wiederholt für zusätzliche (D – 1) Durchläufe verwendet werden zum Erzeugen der verbleibenden D – 1 Komponenten des Ausgangsfarbraumwerts **11**. Oder es könnte zusätzliche (D – 1) Duplikate eines Teils der Hardwareimplementierung geben, die in **Fig.** 30 gezeigt ist, um jede der D Komponenten gleichzeitig zu erzeugen. Die Hardware, die zum Erzeugen von Multiplexersteuereingangssignalen verwendet wird, könnte für jede der D Duplikationen verwendet werden. Der in **Fig.** 30 gezeigte nichtsymmetrische radiale Interpolator **1900** entspricht d = 3 und n = 4, wobei n gleich dem höchsten von n_1, n_2, n_3, ... n_d für den Engangsfarbraumwert **10** ist. Die in **Fig.** 30 gezeigte Hardwareimplementierung implementiert die Gleichungen **14** bis **19** und die Gleichungen der Tabelle 4 für die Erzeugung der Unterwürfelscheitelwerte. Der nichtsymmetrische radiale Interpolator **1900** von **Fig.** 30 ist aus einer ersten, zweiten, dritten und vierten Stufe **1901**–**1904** gebildet. Jede der vier Stufen **1901**–**1904** umfaßt 2^3 Addierer, von denen einer als **1905** gekennzeichnet ist. Jede der vier Stufen **1090**–**1904** umfaßt ferner 2^3 Multiplexer, von denen einer als **1906** gekennzeichnet ist. Schließlich umfaßt jede der vier Stufen **1901**–**1904** 2^3 Steuereingangssignal-Berechnungsblöcke zum Durchführen der Berechnungen der Gleichungen 14 bis 19 mit den angezeigten Eingangssignalen. Einer dieser Steuereingangssignal-Berechnungsblöcke ist als **1907** bezeichnet.

[0156] Jede der Stufen **1901**–**1904** führt eine Iteration der Interpolation durch. Einige Interpolationsanwendungen können erfordern, daß nur eine einzige Iteration der Interpolation durchgeführt wird. Für eine einzige Iteration der Interpolation ist n = 1. Dies entspricht einer Hardwareimplementierung eines nichtsymmetrischen radialen Interpolators **1900** unter Verwendung nur der ersten Stufe **1901**. Eine zusätzliche Stufe würde für jede zusätzliche Iteration der Interpolation addiert, die für die spezielle Anwendung erforderlich ist.

[0157] Jedes der Eingangssignale der Multiplexer in der ersten Stufe **1901** ist mit den acht Scheitelwerten verbunden, die unter Verwendung von Bits höherer Ordnung **10a** ausgewählt werden. Die Ausgangssignale jedes Multiplexers in der ersten Stufe **1901** sind mit dem ersten Eingangssignal des entsprechenden Addierers verbunden. Das zweite Eingangssignal des Addierers ist mit dem Wert verbunden, der der Anzahl von dem Scheitel gleich einem der Eingangssignale der entsprechenden Steuereingangssignal-Berechnungsblöcke entspricht. Das Ausgangssignal von jedem der Addierer der ersten Stufe **1901** ist ein Scheitelwert des ersten Unterwürfels. Wie es vorher erwähnt wurde, wird die Division durch 2 für jede Iteration von Unterwürfelerzeugung verzögert, bis der letzte Unterwürfel erzeugt ist. Die Scheitelwerte des letzten erzeugten Unterwürfels werden durch 2^n dividiert, wobei n der Anzahl von Stufen in dem nichtsymmetrischen radialen Interpolator entspricht. Die Eingangssignale jedes Multiplexers für die zweite, dritte und vierte Stufe **1902**–**1904** sind zu den Ausgangssignalen der Addierer der vorhergehenden Stufe gekoppelt.

[0158] Das Steuereingangssignal jedes Multiplexers ist mit dem Ausgangssignal des entsprechenden Steuereingangssignal-Berechnungsblöcke verbunden. Die Multiplexer, die in dem nichtsymmetrischen radialen Interpolator **1900** verwendet werden, haben die Fähigkeit, unter Verwendung eines 3-Bit-Steuereingangssignals einen von acht 8-Bit-Werten auszuwählen. Für diese d = 3, n = 4 Implementierung des nichtsymmetrischen radialen Interpolators **1900** ist jedes der Eingangssignale in die Steuereingangssignal-Berechnungsblöcke eine 3-Bit-Größe. Das Ausgangssignal von jedem der Steuereingangssignal-Berechnungsblöcke ist eine 3-Bit-Größe. Die Addierer, die jedem Multiplexer zugeordnet sind, führen eine Addition der ausgewählten 8-Bit-Werte von jedem der Multiplexer durch.

[0159] Die Interpolation wird durchgeführt durch Zuführen der Scheitelwerte, die unter Verwendung Bits höherer Ordnung **10a** ausgewählt werden, zu den Multiplexereingängen der ersten Stufe **1901**. Die Eingänge zu den Steuereingangssignal-Berechnungsblöcken sind fest verdrahtet. Der nichtsymmetrische radiale Interpolator **1900** berechet die Scheitelwerte für vier Iterationen der Unterwürfelerzeugung. Die Werte P""(7) bis P""(0) sind die Werte der Scheitel des vierten erzeugten Unterwürfels. Bei diesem Ausführungsbeispiel wird P""(0) ausgewählt, durch 16 dividiert und als das Ergebnis der Interpolation verwendet. Die Teilung durch 16 wird

durch Verschieben von Bits implementiert und ist in **Fig.** 30 nicht dargestellt.

[0160] Ein Durchschnittsfachmann auf diesem Gebiet wird erkennen, daß einer der anderen berechneten Werte P""(7) bis P""(1) ausgewählt werden kann, durch 16 dividiert und als das Ergebnis der Interpolation verwendet werden kann. Das Verwenden von Werten, die anderen Scheiteln des letzten Unterwürfels entsprechen, um das Interpolationsergebnis zu erzeugen, wird das Ergebnis der Interpolation anders vorspannen. Dies ist eine Betrachtung beim Auswählen, welcher Scheitelwert des letzten Unterwürfels, der erzeugt wurde, durch 16 dividiert wird, um das Ergebnis der Interpolation zu erzeugen.

[0161] Die beschnittene vierflächige Interpolation kann für die Implementierung in einem Farbraum angepaßt werden, der durch ein nichtsymmetrisches Würfelgitter dargestellt ist. Wie es für eine beschnittene vierflächige Interpolation der Fall war, werden die Scheitel des erzeugten Unterwürfels unter Verwendung der Scheitel eines Tetraeders berechnet, der von dem Würfel unterteilt ist, der verwendet wird, um den Unterwürfel zu erzeugen.

[0162] Für einige Iterationen der Unterwürfelerzeugung kann es jedoch sein, daß die entsprechenden Bits von jeder der a_l, b_l und c_l-Komponente nicht vorliegen. Für diese Fälle muß die Berechnung der Unterwürfelscheitel modifiziert werden, um die fehlenden entsprechenden Bits bei einer oder mehreren der a_l, b_l und c_l Komponenten auszugleichen.

[0163] In Tabelle 5 und den Gleichungen 25 und 26 sind die Beziehungen gezeigt, die notwendig sind, um die Werte der Unterwürfelscheitel für eine nichtsymmetrische beschnittene vierflächige Unterwürfelerzeugung zu berechnen. Die Gleichungen 25 und 26 zusammen mit den Gleichungen 14 bis 18 werden verwendet, um die richtigen Werte zu erzeugen, so daß bei den in Tabelle 5 aufgelisteten Beziehungen die richtigen Unterwürfelscheitelwerte mit oder ohne die Bits von a_l, b_l und c_l erzeugt werden, die dieser Iteration der vorliegenden Unterwürfelerzeugung entsprechen. Falls für eine gegebene Iteration der Unterwürfelerzeugung ein Bit in einem oder mehreren von a_l, b_l oder c_l nicht vorliegt, erzeugen die Gleichung 25 und die Gleichung 26 die Nummer des Scheitels des Würfels, der beim Erzeugen eines Scheitelwerts des Unterwürfels verwendet wird, so daß eine Kompensation für das/die fehlenden Bits durchgeführt wird.

Gl. 25 $$g(N, i) = (v[i]|\sim m[i] \,\&\, N)$$

Gl. 26 $$h(N, i) = (v[i]|N)$$

Tabelle 5

P'[7] = {P[g(7, i)] + P[h(7, i)]} ÷ 2
P'[6] = {P[g(6, i)] + P[h(6, i)]} ÷ 2
P'[5] = {P[g(5, i)] + P[h(5, i)]} ÷ 2
P'[4] = {P[g(4, i)] + P[h(4, i)]} ÷ 2
P'[3] = {P[g(3, i)] + P[h(3, i)]} ÷ 2
P'[2] = {P[g(2, i)] + P[h(2, i)]} ÷ 2
P'[1) = {P[g(1, i)] + P[h(1, i)]} ÷ 2
P'[0] = {P[g(0, i)] + P[h(0, i)]} ÷ 2

[0164] Obwohl **Fig.** 24 eine graphische Darstellung der Erzeugung eines Unterwürfels **1400** von einem Würfel **1401** unter Verwendung einer nichtsymmetrischen radialen Unterwürfelerzeugung liefert, kann **Fig.** 24 auch eine graphische Darstellung einer nichtsymmetrischen beschnittenen vierflächigen Unterwürfelerzeugung liefern. Bei der speziellen Iteration der in **Fig.** 24 gezeigten Unterwürfelerzeugung liegt das Bit, das der Iteration für die Komponenten b_l und c_l entspricht, nicht vor. Daher geben die Beziehungen in Tabelle 5, wobei g(N, i) und h(N, i), die für die entsprechenden Bits von b_l und c_l berechnet wurden, nicht vorliegen, vor, daß die Scheitelwerte P'[7], P'[6], P'[5] und P'[4] als Mittelwert der Werte der beiden Scheitel berechnet werden, die jeweils mit P'[7], P'[6], P'[5] und P'[4] vertikal ausgerichtet sind. Die Berechnung der Scheitelwerte für andere Kombinationen, bei denen der von a_l, b_l oder c_l vorliegt oder nicht vorliegt, wird analog durch die Gleichungen 25 und 26 gehandhabt.

[0165] In **Fig.** 31 ist eine schematische Darstellung der nichtsymmetrischen beschnittenen vierflächigen Interpolationsberechnung dargestellt. Die Anzahl von Berechnungen, die erforderlich sind, um die nichtsymmetrische beschnittene vierflächige Interpolation durchzuführen, wird berechnet als:

Gl. 27

$$\text{Anzahl von ALU - Operationen} = n \times [(4 \times d) - 1] + 4 \times \left\{2^{n-1} \times \sum_{i=1}^{n-1} \left(1 - 2^{-i}\right)\right\}$$

Gl. 28

$$\text{Anzahl von Speicherzugriffen} = D(\min(2^d, 2^n))$$

[0166] In **Fig.** 32 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer nichtsymmetrischen beschnittenen vierflächigen Interpolation gezeigt. Für dieses Verfahren sind die Eingangsfarbraumwerte **10** aus d Komponenten gebildet. Jede der d Komponenten ist in einen Satz von Bits höherer Ordnung **10a** und Bits niedrigerer Ordnung **10b** unterteilt. Die d Sätze von Bits niedrigerer Ordnung sind jeweils als lb_1, lb_2, lb_3, ... lb_d bezeichnet. Jeder der d Sätze von Bits niedrigerer Ordnung **10b** wird jeweils aus n_1, n_2, n_3, ... n_d Bits gebildet. Die Bitposition von jedem der d Sätze von Bits niedrigerer Ordnung wird bestimmt von dem höchstwertigsten Bit zu dem niedrigstwertigsten Bit durch entsprechende Werte von i_1, i_2, i_3, ... i_a bezeichnet, die jeweils entsprechend von $n_1 - 1$ bis 0, $n_2 - 1$ bis 0, $n_3 - 1$ bis 0, ... $n_d - 1$ bis 0 reichen.

[0167] Zunächst wird ein Satz von $2^n - 2$ Werten berechnet 2000, unter Verwendung von g(N, i) =(v[i]|~m[i] & N) und h(N, i) = (v[i]|N), wobei $m[i] = 2^{d-1} \times Maske_1[i] + 2^{d-2} \times Maske_2[i] + 2^{d-3} \times Maske_3[i] + ... + 2^{d-d} \times Maske_d[i]$. Für diese Berechnung sind die Werte der $Maske_j = 2^k - 1$ jeweils für einen Wert k berechnet, der aus n_1, n_2, n_3, ... n_d ausgewählt wird, wobei der Wert von j dem Wert des tiefergestellten Zeichens des ausgewählten von n_1, n_2, n_3, ..., n_d entspricht. Die Werte von j reichen von 1 bis d. Die Werte von v[i] werden berechnet als m[i] & $(2^{d-1} \times lb_1[i] + 2^{d-2} \times lb_2[i] + 2^{d-3} \times lb_3[i] + ... + 2^{d-d} \times lb_d[i])$ für Werte von i, die von n – 1 bis 0 reichen, wobei n dem größten von n_1, n_2, n_3, ..., n_d entspricht. Der Wert von N entspricht den Scheitelnummern, die von 1 bis 2^d reichen. Nachfolgend wird eine Anzahl von Interpolationsdatenwerten gleich dem Minimum von 2^n und 2^d ausgewählt **2001** unter Verwendung der eindeutigen Werte in dem Satz von $2^n - 2$ Werten, einem Wert von v[i] für i in dem Bereich von n – 1 bis 0, und den d Sätzen von Bits höherer Ordnung. Schließlich werden die ausgewählten Interpolationsdatenwerte addiert **2002**, um eine Summe zu erzeugen. Um einen Abrundungsfehler zu vermeiden, wird die erzeugte Summe durch 2^n dividiert. Dieser Schritt ist in **Fig.** 32 nicht gezeigt.

[0168] In **Fig.** 33 ist ein Flußdiagramm hoher Ebene eines Verfahrens gezeigt, das in Software implementiert ist, um eine nichtsymmetrische beschnittene vierflächige Interpolation durchzuführen. Zunächst werden die Maskenwerte für jede Komponente des Eingangsfarbraumwerts **10** erzeugt **2100**. Nachfolgend wird eine Bestimmung **2101** durchgeführt, ob irgendeine der Komponenten des Eingangsfarbraumwerts (a, b, c) **10** einer Position an einer äußeren Grenze des Würfelgitters 1 entspricht. Dies ist der Fall, falls eine oder mehrere der Komponenten des Eingangsfarbraumwerts einen hexadezimalen Wert von FF aufweist, Falls dies der Fall ist, wird zum Zweck des Erzeugens des Index in das Würfelgitter 1 zum Wiedergewinnen der notwendigen Scheitelwerte, den Komponenten des Eingangsfarbraumwerts **10**, die einen hexadezimalen Wert von FF aufweisen, ein hexadezimaler Wert von **100** zugewiesen **2102**. Dann werden die Werte von m[i] und v[i] berechnet **2103**. Nachfolgend werden die Indizes, die verwendet werden, um auf jeden der Scheitelwerte zuzugreifen, die für die Interpolation verwendet werden, unter Verwendung von g(N, i) und h(N, i) berechnet **2104**. Schließlich wird jede der Komponenten des Ausgangsfarbraumwerts unter Verwendung der Werte berechnet **2105**, auf die durch die Indizes zugegriffen wird, die im Schritt **2104** berechnet werden.

[0169] In **Fig.** 45 ist eine Auflistung des Codes für eine Softwareimplementierung einer nichtsymmetrischen beschnittenen vierflächigen Interpolation in C enthalten. Für die nichtsymmetrische beschnittene vierflächige Interpolation entsprechen die berechneten Indizes Versätzen von dem Ursprung des Würfels, der durch die Bits höherer Ordnung **10a** ausgewählt wird. Aufgrund der sich ändernden Auflösung, die in dem gesamten Ausgangsfarbraum verwendet wird, werden die Werte der Scheitel für die drei Würfel (ein Würfel für jede Abmessung des Ausgangsfarbraums), die durch die Bits höherer Ordnung **10a** ausgewählt werden, in die Routine von **Fig.** 45 geleitet, jedesmal wenn an einem Eingangsfarbraumwert **10** eine Farbraumumwandlung durchgeführt wird. Dies unterscheidet sich von dem Code für die beschnittene radiale und beschnittene vierflächige Interpolation, bei der die Farbtabelle als ein Array in die Routine geleitet wird und Indizes in diese Tabelle in der Routine berechnet werden.

[0170] Es ist auch möglich, eine nichtsymmetrische beschnittene vierflächige Interpolation in Hardware zu implementieren. Wie es vorher erwähnt wurde, werden Verschiebungen und Verkettungen implementiert, ohne zusätzliche Hardwareelemente zu erfordern. Die Hardwarefunktionsblöcke, die erforderlich sind, um die nichtsymmetrische beschnittene vierflächige Interpolation durchzuführen, umfassen Addierer, UND-Gatter, ODER-Gatter und Multiplexer. Mit D Abmessungen in dem Ausgangsfarbraum, d Abmessungen in dem Eingangsfarbraum und n Bits, die die maximale Anzahl von Bits darstellen, die verwendet werden, um eine der

Komponenten des Eingangsfarbraums darzustellen, können die Anforderungen der Hardwareimplementierung der nichtsymmetrischen beschnittenen vierflächigen Interpolation berechnet werden als:

Gl. 29 $$\text{Anzahl von Addierern} = D \times 2^n$$

Gl. 30 $$\text{Anzahl von Multiplexern} = D \times (2^n - 1)$$

Gl. 31 $$\text{Anzahl von zusätzlichen UND/ODER - Gattern} = (4 \times d) \times \left\{ 2^n - 1 \times \sum_{i=1}^{n-1} \left(1 - 2^{-i}\right) \right\}$$

[0171] Um den Gatterpegelentwurf zu erzeugen, der notwendig ist, um die nichtsymmetrische beschnittene vierflächige Interpolation in Hardware zu implementieren, kann eine üblicherweise verwendete Hardwarebeschreibungssprache, wie z. B. VHDL, verwendet werden. In **Fig.** 46 ist eine Auflistung des VHDL-Codes enthalten, der eine Hardwareimplementierung einer nichtsymmetrischen beschnittenen vierflächigen Interpolation erzeugen kann.

[0172] In **Fig.** 34 ist eine Hardwareimplementierung eines nichtsymmetrischen beschnittenen vierflächigen Interpolators **2200** gezeigt. Die Steuereingangssignal-Berechnungsblöcke **2201a**–**2201v** berechnen die Werte, die durch die Steuereingangssignale der Multiplexer **2202a**–**2202n** verwendet werden. Die Steuereingangssignal-Berechnungsblöcke **2201a**–**2201v** legen die Funktionen der Gleichungen 25 und 26 an, wie es in **Fig.** 34 angezeigt ist, um die Steuereingangssignale für die Multiplexer **2202a**–**2202n** zu berechnen. Wie es in **Fig.** 34 gezeigt ist, verwendet der Multiplexer **2202o** einen Wert von v[i] für sein Steuereingangssignal. Jeder der Multiplexer **2202a**–**2202o** wählt aus acht Interpolationsdatenwerten aus, die unter Verwendung von Bits höherer Ordnung **10a** ausgewählt wurden. Die Addierer **2203a**–**2203o** summieren die Ausgangssignale der Multiplexer **2202a**–**2202o**. Diese Summe wird durch Bitverschieben (in **Fig.** 34 nicht gezeigt) durch 2^n dividiert, um das Interpolationsergebnis zu erzeugen.

[0173] Die Hardware von **Fig.** 28 könnte wiederholt für zusätzliche (D – 1) Durchläufe verwendet werden, um die verbleibenden D – 1 Komponenten des Ausgangsfarbraumwerts **11** zu erzeugen. Oder es könnte zusätzliche (D – 1) Duplikationen von einem Teil der in **Fig.** 28 gezeigten Hardwareimplementierung geben, um jede der D Komponenten gleichzeitig zu erzeugen. Die Hardware, die zum Erzeugen von Multiplexersteuereingangssignalen verwendet wird, könnte für jede der D Duplikationen verwendet werden. Der in **Fig.** 34 gezeigte nichtsymmetrische beschnittene vierflächige Interpolator **2200** entspricht d = 3 und dem Maximum von einem von n_1, n_2, n_3, ... n_d gleich 4 für den Eingangsfarbraumwert **10**. Der nichtsymmetrische beschnittene vierflächige Interpolator **2200** implementiert die schematische Darstellung der in **Fig.** 31 gezeigten nichtsymmetrischen beschnittenen vierflächigen Interpolation.

[0174] In **Fig.** 35 ist ein Flußdiagramm hoher Ebene eines verallgemeinerten Verfahrens zum Durchführen einer nichtsymmetrischen vierflächigen Interpolation gezeigt. Zunächst wird ein erster und ein zweiter Satz von 2^d Werten unter Verwendung von g(N, i) bzw. h(N, i) berechnet **2300**. Danach werden 2^d Paare von Interpolationsdatenwerten unter Verwendung des ersten und des zweiten Satzes von Werten ausgewählt **2301**. Schließlich wird ein Satz von 2^d Summen unter Verwendung der 2^d Paare von Interpolationsdatenwerten berechnet **2302**.

[0175] Das in **Fig.** 35 gezeigte Verfahren ist für eine einzige Iteration einer nichtsymmetrischen vierflächigen Interpolation. Es sollte klar sein, daß weitere Iterationen durchgeführt würden durch Wiederholen der Schritte von **Fig.** 35, mit den aufeinanderfolgenden Sätzen von 2^d Werten, die unter Verwendung von g(N, i) und h(N, i) berechnet wurden, für Werte von v[i] und m[i], die auf einanderfolgend dekrementierten Werten von i entsprechen, Auswählen aufeinanderfolgender Sätze von 2^d Paaren von Werten von dem vorher berechneten Satz von 2^d Summen, und Berechnen aufeinanderfolgender Sätze von 2^d Summen von den aufeinanderfolgenden Sätzen von 2^d Paaren von Werten. Nach. dem Durchführen von n Iterationen, wo n gleich dem höchsten von n_1, n_2, n_3, ... n_d ist, wird eine der 2^d Summen des letzten berechneten Satzes durch 2^n dividiert (in **Fig.** 35 nicht gezeigt), um das Ergebnis zu erzeugen. Die Division durch 2, die nach jeder Iteration durchgeführt werden könnte, wird bis nach der letzten Iteration verzögert, um Abrundungsfehler zu vermeiden.

[0176] In **Fig.** 36 ist eine Hardwareimplementierung eines nichtsymmetrischen vierflächigen Interpolators **2400** für die Umwandlung von Eingangsfarbraumwerten **10** zu einer Komponente von Ausgangsfarbraumwerten **11** gezeigt. Diese gleiche Hardware könnte wiederholt für zusätzliche (D – 1) Durchläufe verwendet wer-

den, um die verbleibenden D – 1. Komponenten des Ausgangsfarbraumwerts **11** zu erzeugen. Oder es könnte zusätzliche (D – 1) Duplikationen von einem Teil der in **Fig.** 36 gezeigten Hardwareimplementierung geben, um jede der D Komponenten gleichzeitig zu erzeugen. Die Hardware, die zum Erzeugen von Multiplexersteuereingangssignalen verwendet wird, könnte für jede der D Duplikationen verwendet werden. Der in **Fig.** 36 gezeigte nichtsymmetrische vierflächige Interpolator **2400** entspricht d = 3 und n = 4, wobei n gleich dem höchsten von n_1, n_2, n_3, ... n_d für den Eingangsfarbraumwert **10** ist. Die in **Fig.** 36 gezeigte Hardwareimplementierung implementiert die Gleichungen **25** und **26** und die Gleichungen von Tabelle 5 für die Erzeugung der Unterwürfelscheitelwerte. Der nichtsymmetrische vierflächige Interpolator **2400** von **Fig.** 36 ist aus einer ersten, zweiten, dritten und vierten Stufe **2401**–**2404** gebildet. Jede der vier Stufen **2401**–**2404** umfaßt 2^3 Addierer, von denen einer als **2405** bezeichnet ist. Jede der vier Stufen **1901**–**1904** umfaßt ferner 2×2^3 Multiplexer, von denen einer als **2406** bezeichnet ist. Die Multiplexer für jede Stufe sind in 2^3 Paaren angeordnet. Schließlich umfaßt jede der vier Stufen **2401**–**2404** 2^3 Steuereingangssignal-Berechnungsblöcke zum Berechnen von g(N, i) und 2^3 Steuereingangssignal-Berechnungsblöcke zum Berechnen von h(N, i). Einer der Steuereingangssignal-Berechnungsblöcke zum Berechnen von g(N, i) ist als **2407** bezeichnet und einer der Steuereingangssignal-Berechnungsblöcke zum Berechnen von h(N, i) ist als **2408** bezeichnet.

[0177] Jede der Stufen **2401**–**2404** führt eine Iteration der Interpolation durch. Einige Interpolationsanwendungen können erfordern, daß nur eine einzige Iteration der Interpolation durchgeführt wird. Für eine einzige Iteration der Interpolation ist n = 1. Dies entspricht einer Hardwareimplementierung eines nichtsymmetrischen vierflächigen Interpolators **2400** unter Verwendung nur der ersten Stufe **2401**. Eine zusätzliche Stufe würde für jede zusätzliche Iteration der Interpolation addiert, die für die spezielle Anwendung erforderlich ist.

[0178] Jedes der Eingangssignale der Multiplexer in der ersten Stufe **2401** ist mit den acht Scheitelwerten verbunden, die unter Verwendung der Bits höherer Ordnung **10a** ausgewählt werden. Die Ausgangssignale jedes Paars von Multiplexern in den Stufen **2401**–**2404** sind jeweils mit dem ersten und dem zweiten Eingangssignal des entsprechenden Addierers verbunden. Das Ausgangssignal von jedem der Addierer der ersten Stufe **1901** ist ein Scheitelwert des ersten Unterwürfels. Wie es vorher erwähnt wurde, wird die Division durch 2 für jede Iteration der Unterwürfelerzeugung verzögert, bis der letzte Unterwürfel erzeugt ist. Die Scheitelwerte des letzten erzeugten Unterwürfels werden durch 2^n dividiert (in **Fig.** 36 nicht gezeigt), wobei n der Anzahl von Stufen in dem nichtsymmetrischen vierflächigen Interpolator entspricht. Die Eingänge jedes Multiplexers für die zweite, dritte und vierte Stufe **2402**–**2404** sind zu den Ausgängen der Addierer der vorhergehenden Stufe gekoppelt.

[0179] Das Steuereingangssignal jedes Multiplexers ist mit dem Ausgangssignal des entsprechenden Steuereingangssignal-Berechnungsblocks verbunden. Die in dem nichtsymmetrischen vierflächigen Interpolator **2400** verwendeten Multiplexer haben die Fähigkeit, unter Verwendung eines 3-Bit-Steuereingangssignals einen von acht 8-Bit-Werten auszuwählen. Für diese d = 3 Implementierung eines nichtsymmetrischen vierflächigen Interpolators **2400** ist jedes der Eingangssignale in die Steuereingangssignal-Berechnungsblöcke eine 3-Bit-Größe. Das Ausgangssignal von jedem der Steuereingangssignal-Berechnungsblöcke ist eine 3-Bit-Größe. Die Addierer, die jedem Multiplexer zugeordnet sind, führen eine Addition der ausgewählten 8-Bit-Werte von jedem der Multiplexer durch.

[0180] Die Interpolation wird durchgeführt durch Zuführen der Scheitelwerte, die unter Verwendung Bits höherer Ordnung **10a** ausgewählt werden, zu den Multiplexereingängen der ersten Stufe **2401**. Die Eingänge zu den Steuereingangssignal-Berechnungsblöcken sind fest verdrahtet. Der nichtsymmetrische vierflächige Interpolator **2400** berechnet die Scheitelwerte für vier Iterationen der Unterwürfelerzeugung. Die Werte P""(7) bis P""(0) sind die Werte der Scheitel des vierten erzeugten Unterwürfels. Bei diesem Ausführungsbeispiel wird P""(0) ausgewählt, durch 16 dividiert und als das Ergebnis der Interpolation verwendet. Die Division durch 16 wird durch Verschieben von Bits implementiert und ist in **Fig.** 36 nicht dargestellt.

[0181] Ein Durchschnittsfachmann auf diesem Gebiet erkennt, daß einer der anderen berechneten Werte P""(7) bis P""(1) ausgewählt werden kann, durch 16 dividiert und als das Ergebnis der Interpolation verwendet werden kann. Das Verwenden von Werten, die anderen Scheiteln des letzten Unterwürfels entsprechen, um das Interpolationsergebnis zu erzeugen, wird das Ergebnis der Interpolation anders vorspannen. Dies ist eine Überlegung beim Auswählen, welcher Scheitelwert des letzten erzeugten Unterwürfels durch 16 dividiert wird, um das Ergebnis der Interpolation zu erzeugen.

[0182] Eine gemeinsame Hardwareimplementierung von nichtsymmetrischer radialer Interpolation und nichtsymmetrischer beschnittener vierflächiger Interpolation ist möglich. Wie es von den schematischen Darstellungen der nichtsymmetrischen radialen Interpolation und der nichtsymmetrischen beschnittenen vierflächigen In-

terpolation in **Fig.** 25 bzw. **Fig.** 31 ersichtlich ist, könnte eine gemeinsame Hardwareimplementierung erreicht werden durch Multiplexen der Indizes, die verwendet werden, um auf die Eingangsscheitelwerte zuzugreifen. In **Fig.** 47 ist eine Auflistung des VHDL-Codes enthalten, der eine gemeinsame Hardwareimplementierung einer nichtsymmetrischen radialen und nichtsymmetrischen beschnittenen vierflächigen Interpolation erzeugen kann.

[0183] In **Fig.** 37 ist eine Hardwareimplementierung eines gemeinsamen nichtsymmetrischen beschnittenen radialen und nichtsymmetrischen beschnittenen vierflächigen Interpolators **2500** gezeigt. Die Hardwareimplementierung der gemeinsamen nichtsymmetrischen beschnittenen radialen und nichtsymmetrischen beschnittenen vierflächigen Interpolation umfaßt die Steuereingangssignal-Berechnungsblöcke des nichtsymmetrischen beschnittenen radialen **1700** und des nichtsymmetrischen beschnittenen vierflächigen **2200** Interpolators. Die Multiplexer **2501a**–**2501n** werden verwendet, um die Daten an den Steuereingängen der Multiplexer **2502a**–**2502n** auszuwählen. Ein einzelnes Bit wird verwendet, um die Auswahl der Daten an den Eingängen der Multiplexer **2501a**–**2501n** zu steuern. Das einzelne Bit steuert, ob die Multiplexersteuereingangssignale in die Multiplexer **2502a**–**2502n** für eine nichtsymmetrische beschnittene vierflächige Interpolation oder für eine nichtsymmetrische beschnittene radiale Interpolation sind. Das Multiplexersteuereingangssignal für die Multiplexer **2502a**–**2502n** bestimmt, welcher der Interpolationsdatenwerte mit den Addierern gekoppelt wird. Wenn das Bit in dem ersten seiner zwei Zustände ist, arbeitet die Hardware von **Fig.** 37 als ein nichtsymmetrischer beschnittener radialer Interpolator. Mit dem Bit in dem zweiten seiner zwei Zustände arbeitet die Hardware von **Fig.** 37 als ein nichtsymmetrischer beschnittener vierflächiger Interpolator. Die Addierer **2503a**–**2503o** summieren die Ausgangssignale der Multiplexer **2502a**–**2502o**. Durch Verschieben von Bits wird die resultierende Summe durch 16 dividiert (in **Fig.** 22 nicht gezeigt), um das Ergebnis zu erzeugen. Die Steuereingangssignal-Berechnungsblöcke **2504a**–**2504k** implementieren die Gleichungen 14 bis 19, und die Steuereingangssignal-Berechnungsblöcke **2505a**–**2505v** implementieren die Gleichungen 25 und 26.

[0184] Die Hardware von **Fig.** 37 könnte wiederholt für zusätzliche (D – 1) Durchläufe verwendet werden, um die verbleibenden D – 1 Komponenten des Ausgangsfarbraumwerts **11** zu erzeugen. Oder es könnte zusätzliche (D – 1) Duplikationen eines Teils der in **Fig.** 37 gezeigten Hardwareimplementierung geben, um jede der D Komponenten gleichzeitig zu erzeugen. Die Hardware, die zum Erzeugen von Multiplexersteuereingangssignalen verwendet wird, könnte für jede der D Duplikationen verwendet werden. Der in **Fig.** 37 gezeigte gemeinsame nichtsymmetrische beschnittene radiale und nichtsymmetrische beschnittene vierflächige Interpolator **2500** entspricht d = 3 und n = 4 für den Eingangsfarbraumwert **10**.

[0185] Es sollte klar sein, daß für jedes der offenbarten Hardwareausführungsbeispiele von Interpolationen Berechnungen erforderlich sind, um die Multiplexersteuereingangssignale zu liefern. Diese Berechnungen können in speziell zugewiesener Hardware implementiert werden, oder unter Verwendung eines Mikroprozessors unter Softwaresteuerung durchgeführt werden. Das Verwenden eines Mikroprozessors zum Berechnen der Multiplexersteuereingangssignale führt zu Hardwareeinsparungen auf Kosten des Anstiegs der Zeit, die erforderlich ist, um die Multiplexersteuereingangsberechnungen durchzuführen.

[0186] Obwohl mehrere Ausführungsbeispiele der Erfindung dargestellt wurden und die Formen derselben beschrieben wurden, ist es für einen Durchschnittsfachmann auf diesem Gebiet ohne weiteres offensichtlich, daß verschiedene Modifikationen darin durchgeführt werden können, ohne von dem Schutzbereich der angehängten Ansprüche abzuweichen.

Patentansprüche

1. Ein gemeinsamer nichtsymmetrischer beschnittener radialer und nichtsymmetrischer beschnittener vierflächiger Interpolator (**2500**) zum Interpolieren zwischen Interpolationsdatenwerten unter Verwendung von Eingangsdatenwerten (**10**), die jeweils d Komponenten aufweisen, um Ausgangsdatenwerte (**11**) zu erzeugen, wobei die d Komponenten durch d Sätze von Bits entsprechend dargestellt sind, die jeweils unterteilt sind, um d Sätze von Bits niedrigerer Ordnung (**10b**) zu bilden, die jeweils n_1, n_2, n_3, ... n_d der Bits aufweisen, wobei der gemeinsame nichtsymmetrische beschnittene radiale und nichtsymmetrische beschnittene vierflächige Interpolator folgende Merkmale umfaßt:
einen ersten Satz von 2^n – 1 Multiplexern (**2502a**–**2502o**), die jeweils konfiguriert sind zum Empfangen eines von einem ersten Satz von Steuereingangssignalen und einen Multiplexerausgang aufweisen, mit n gleich dem Größten von n_1, n_2, n_3 ... n_d, wobei jeder der Multiplexer (**2502a**–**2502o**) des ersten Satzes vorgesehen ist, um ansprechend auf den einen des ersten Satzes von Steuereingangssignalen aus den Interpolationsdatenwerten auszuwählen;
einen zweiten Satz von 2^n – 2 Multiplexern (**2501a**–**2501n**), die jeweils einen Multiplexerausgang aufweisen,

wobei jeder der Multiplexerausgänge des zweiten Satzes mit einem der Multiplexer (**2502a**–**2502o**) des ersten Satzes von Multiplexern (**2502a**–**2502o**) gekoppelt ist, wobei der zweite Satz von Multiplexern (**2501a**–**2501n**) vorgesehen ist, um $2^n - 2$ des ersten Satzes von Steuereingangssignalen durch jeden des zweiten Satzes von Multiplexern (**2501a**–**2501o**) zuzuführen, die ansprechend auf einen von einem zweiten Satz von Steuereingangssignalen zwischen einem von einem ersten Satz von Werten und einem von einem zweiten Satz von Werten auswählen, wobei. das Bestimmen des ersten Satzes von Werten das Verwenden einer ersten Funktion umfaßt, die eine Bitmaske verwendet und eine Binärzahl verwendet, die durch Auswählen des höchstwertigsten Bits von jedem der Eingangsdatenwerte und durch Verketten der höchstwertigsten Bits in einen binären Wert berechnet wird, und wobei das Bestimmen des zweiten Satzes von Werten das Verwenden einer zweiten Funktion, die die Bitmaske und die Binärzahl verwendet, die sich aber von der ersten Funktion unterscheidet, und einer dritten Funktion umfaßt, die die Binärzahl ohne die Bitmaske verwendet, und
eine Einrichtung zum Addieren (**2503a**–**2503o**), die zum Empfangen des Multiplexerausgangssignals von dem ersten Satz von Multiplexern (**2502a**–**2502o**) konfiguriert ist.

2. Der gemeinsame nichtsymmetrisch beschnittene radiale und nichtsymmetrisch beschnittene vierflächige Interpolator (**2500**) gemäß Anspruch 1, der ferner folgende Merkmale umfaßt:
einen ersten Satz von $2^n - n - 1$ Steuereingangsberechnungsblöcken (**2504a**–**2504k**), wobei jeder der Steuereingangsberechnungsblöcke (**2504a**–**2504k**) des ersten Satzes mit einem der Multiplexer (**2501a**–**2501n**) des zweiten Satzes gekoppelt ist, wobei der erste Satz von Steuereingangsberechnungsblöcken (**2504a**–**2504k**) zum Verwenden der ersten Funktion zum Erzeugen von $2^n - n - 1$ des ersten Satzes von Werten vorgesehen ist; und
einen zweiten Satz von $2^n - 2$ Steuereingangsberechnungsblöcken (**2505a**–**2505n**), wobei jeder der Steuereingangsberechnungsblöcke (**2505a**–**2505n**) des zweiten Satzes mit einem der Multiplexer (**2501a**–**2501n**) des zweiten Satzes gekoppelt ist, wobei der zweite Satz von Steuereingangsberechnungsblöcken (**2505a**–**2505n**) zum Verwenden der zweiten Funktion und der dritten Funktion zum Erzeugen des zweiten Satzes von Werten vorgesehen ist.

3. Der gemeinsame nichtsymmetrisch beschnittene radiale und nichtsymmetrisch beschnittene vierflächige Interpolator (**2500**) gemäß Anspruch 2, bei dem:
jeder der Multiplexer (**2502a**–**2502o**) des ersten Satzes 2^d Multiplexereingänge zum Empfangen von 2^d der Interpolationsdatenwerte umfaßt;
jeder der Multiplexer (**2501a**–**2501n**) des zweiten Satzes einen ersten Multiplexereingang und einen zweiten Multiplexereingang umfaßt, die konfiguriert sind, um den einen des ersten Satzes von Werten bzw. den einen des zweiten Satzes von Werten zu empfangen; und
jeder der Multiplexer (**2501a**–**2501n**) des zweiten Satzes die Fähigkeit zum Verwenden des einen des zweiten Satzes von Steuereingangssignalen umfaßt, zum Auswählen zwischen dem Durchführen einer nichtsymmetrisch beschnittenen radialen Interpolation und einer nichtsymmetrischen beschnittenen vierflächigen Interpolation.

4. Der gemeinsame nichtsymmetrisch beschnittene radiale und nichtsymmetrisch beschnittene vierflächige Interpolator (**2500**) gemäß Anspruch 3, bei dem:
lb_1, lb_2, lb_3, ..., lb_d die d Sätze von Bits niedrigerer Ordnung (**10b**) bezeichnen, wobei eine Bitposition jedes Bits der d Sätze von Bits niedrigerer Ordnung (**10b**) von einem höchstwertigsten der Bits niedrigerer Ordnung (**10b**) zu einem niedrigstwertigen der Bits niedrigerer Ordnung (**10b**) durch entsprechende Werte von i_1, i_2, i_3, ... i_d bezeichnet ist, die jeweils entsprechend von $n_1 - 1$ bis 0, $n_2 - 1$ bis 0, $n_3 - 1$ bis 0, ... $n_d - 1$ bis 0 reichen;
die Berechnung des zweiten Satzes von Werten die Verwendung von g (N, i) = (v[i]|~m[i]) & N und h (N, i) = v[i]N für die zweite Funktion bzw. die dritte Funktion umfaßt, mit $m[i] = 2^{d-1} \times Maske_1[i] + 2^{d-2} \times Maske_2[i] + 2^{d-3} \times Maske_3[i] + ... + 2^{d-d} \times Maske_d[i]$, mit jeder $Maske_j = 2^k - 1$ für einen Wert von k, der aus n_1, n_2, n_3, ... n_d ausgewählt ist, und für j, der von 1 bis d reicht, mit $v[i] = m[i] \& (2^{d-1} \times lb_1[i] + 2^{d-2} \times lb_2[i] + 2^{d-3} \times lb_3[i] + ... + 2^{d-d} \times lb_d[i])$ für Werte von i, die von n – 1 bis 0 reichen, und wobei N Werte umfaßt, die von 1 bis 2^d reichen;
die Berechnung von $2^n - n - 1$ des ersten Satzes von Werten die Verwendung von f (N, i) = v[i]|(N&~m[i] für die erste Funktion umfaßt; und
die Berechnung von n – 1 des ersten Satzes von Werten die Verwendung von v[i] für die Werte von i umfaßt, die von n – 2 bis 0 reichen; und
die Berechnung von einem des ersten Satzes von Steuereingängen das Verwenden von v[i] für einen der Werte von i gleich n – 1 umfaßt.

5. Der gemeinsame nichtsymmetrische beschnittene radiale und nichtsymmetrische beschnittene vierflächige Interpolator (**2500**) gemäß Anspruch 4, bei dem:
die Einrichtung zum Addieren (**2503a**–**2503o**) $2^n - i$ Addierer (**2503a**–**2503o**) umfaßt, die jeweils einen ersten

Eingang, einen zweiten Eingang und einen Ausgang umfassen;
$2^{n-1} - 1$ der Addierer (**2503a**–**3503o**) jeweils den ersten und den zweiten Eingang mit einem der Multiplexerausgänge der Multiplexer (**2502a**–**2502o**) des ersten Satzes gekoppelt haben;
der erste Eingang von einem der Addierer (**2503a**–**2503o**) eine Konfiguration zum Empfangen von einem der Interpolationsdatenwerte umfaßt, die unter Verwendung von d Sätzen von Bits höherer Ordnung (**10a**) ausgewählt wurden, die von den d Sätzen von Bits abgeteilt werden, und wobei der zweite Eingang mit einem der Multiplexerausgänge der Multiplexer (**2501a**–**2501n**) des zweiten Satzes gekoppelt ist; und
$2^n - (2^{n-1} - 1)$ der Addierer (**2503a**–**2503o**) jeweils den ersten Eingang und den zweiten Eingang mit dem Ausgang eines anderen der Addierer (**2503a**–**2503o**) gekoppelt haben.

6. Der gemeinsame nichtsymmetrische beschnittene radiale und nichtsymmetrische beschnittene vierflächige Interpolator (**2500**) gemäß Anspruch 5, bei dem:
jeder der Interpolationsdatenwerte D Interpolationsdatenwertkomponenten umfaßt; und
jede der D Interpolationsdatenwertkomponenten einer Abmessung eines Ausgangsfarbraums entspricht, und jede der d Komponenten der Eingangsdatenwerte (**10**) einer Dimension eines Eingangsfarbraums entspricht.

7. Der gemeinsame nichtsymmetrisch beschnittene radiale und nichtsymmetrisch beschnittene vierflächige Interpolator (**2500**) gemäß Anspruch 6, bei dem:
ein Druckgerät den gemeinsamen nichtsymmetrisch beschnittenen radialen und nichtsymmetrisch beschnittenen vierflächigen Interpolator (**2500**) zum Durchführen einer Farbraumumwandlung umfaßt.

8. Der gemeinsame nichtsymmetrisch beschnittene radiale und nichtsymmetrisch beschnittene vierflächige Interpolator (**2500**) gemäß Anspruch 7, bei dem:
D gleich 3 ist, d gleich 3 ist und n gleich 4 ist, wobei der Ausgangsfarbraum einen Farbraum umfaßt, der aus der Gruppe ausgewählt ist, die aus einem RGB-, einem Lab-, einem XYZ-, einem HSV-, einem Luv-, einem HLS- und einem CMY-Farbraum besteht, und der Eingangsfarbraum einen Farbraum umfaßt, der aus der Gruppe ausgewählt ist, die aus einem RGB-, einem Lab-, einem XYZ-, einem HSV-, einem Luv-, einem HLS- und einem CMY-Farbraum besteht.

9. Der gemeinsame nichtsymmetrisch beschnittene radiale und nichtsymmetrisch beschnittene vierflächige Interpolator (**2500**) gemäß Anspruch 7, bei dem:
D gleich 4 ist, d gleich 3 ist und n gleich 4 ist, wobei der Farbraum einen CMYK-Farbraum umfaßt und der Eingangsfarbraum einen Farbraum umfaßt, der von der Gruppe ausgewählt ist, die aus einem RGB-, einem Lab-, einem XYZ-, einem HSV-, einem Luv-, einem HLS- und einem CMY-Farbraum besteht.

10. Ein gemeinsamer nichtsymmetrisch beschnittener radialer und nichtsymmetrisch beschnittener vierflächiger Interpolator (**2500**) zum Interpolieren zwischen Interpolationsdatenwerten unter Verwendung von Eingangsdatenwerten (**10**), die jeweils drei Komponenten aufweisen, um Ausgangsdatenwerte (**11**) zu erzeugen, wobei die Komponenten, die durch drei Sätze von Bits dargestellt sind, die jeweils unterteilt sind, um drei Sätze von Bits niedrigerer Ordnung (**10b**) , die jeweils n_1, n_2, n_3 der Bits aufweisen, und drei Sätze von Bits höherer Ordnung (**10a**) zu bilden, wobei der gemeinsame nichtsymmetrisch beschnittene radiale und nichtsymmetrisch beschnittene vierflächige Interpolator (**2500**) folgende Merkmale umfaßt:
einen ersten Multiplexer (**2502o**) mit einem ersten Multiplexerausgang, der konfiguriert ist zum Empfangen eines ersten Steuereingangssignals und der Interpolationsdatenwerte, wobei der erste Multiplexer (**2502o**) vorgesehen ist, um ansprechend auf das erste Steuereingangssignal aus den Interpolationsdatenwerten auszuwählen;
einen zweiten Multiplexer (**2502n**) mit einem zweiten Multiplexerausgang, der konfiguriert ist zum Empfangen eines zweiten Steuereingangssignals und der Interpolationsdatenwerte, wobei der zweite Multiplexer (**2502n**) vorgesehen ist, um ansprechend auf das zweite Steuereingangssignal aus den Interpolationsdatenwerten auszuwählen;
einen dritten Multiplexer (**2502m**) mit einem dritten Multiplexerausgang, der konfiguriert ist zum Empfangen eines dritten Steuereingangssignals und der Interpolationsdatenwerte, wobei der dritte Multiplexer (**2502m**) vorgesehen ist, um ansprechend auf das dritte Steuereingangssignal aus den Interpolationsdatenwerten auszuwählen;
einen vierten Multiplexer (**2501n**) mit einem vierten Multiplexerausgang, der mit dem zweiten Multiplexer (**2502n**) gekoppelt ist, um das zweite Steuereingangssignal zuzuführen, wobei der vierte Multiplexer (**2501n**) vorgesehen ist, um aus einem ersten Wert, der von einer ersten Funktion bestimmt wird, und einem zweiten Wert, der von einer zweiten Funktion und einer dritten Funktion bestimmt wird, auszuwählen;
einen fünften Multiplexer (**2501m**) mit einem fünften Multiplexerausgang, der mit dem dritten Multiplexer (2502m) gekoppelt ist, um das dritte Steuereingangssignal zuzuführen, wobei der fünfte Multiplexer (**2501m**)

vorgesehen ist, um aus einem dritten Wert, der aus der ersten Funktion bestimmt wird, die eine Bitmaske und eine Binärzahl verwendet, die durch Auswählen des höchstwertigsten Bits von jedem der Ausgangsdatenwerten, und durch Verketten der höchstwertigsten Bits in einen binären Wert berechnet wird, und einem vierten Wert, der aus der zweiten Funktion bestimmt wird, die die Bitmaske und die Binärzahl verwendet, die sich aber von der ersten Funktion unterscheidet, und aus der dritten Funktion, die die Binärzahl ohne die Bitmaske verwendet, auszuwählen; und

eine Einrichtung zum Addieren (**2503g**, **2503h**, **2503l**), die mit dem Ausgang des ersten (**2502o**), des zweiten (2502n) und des dritten Multiplexers (**2502m**) gekoppelt ist, und konfiguriert ist zum Empfangen von einem der Interpolationsdatenwerte, die von den drei Sätzen von Bits höherer Ordnung (**10a**) bestimmt sind.

Es folgen 67 Blatt Zeichnungen

Anhängende Zeichnungen

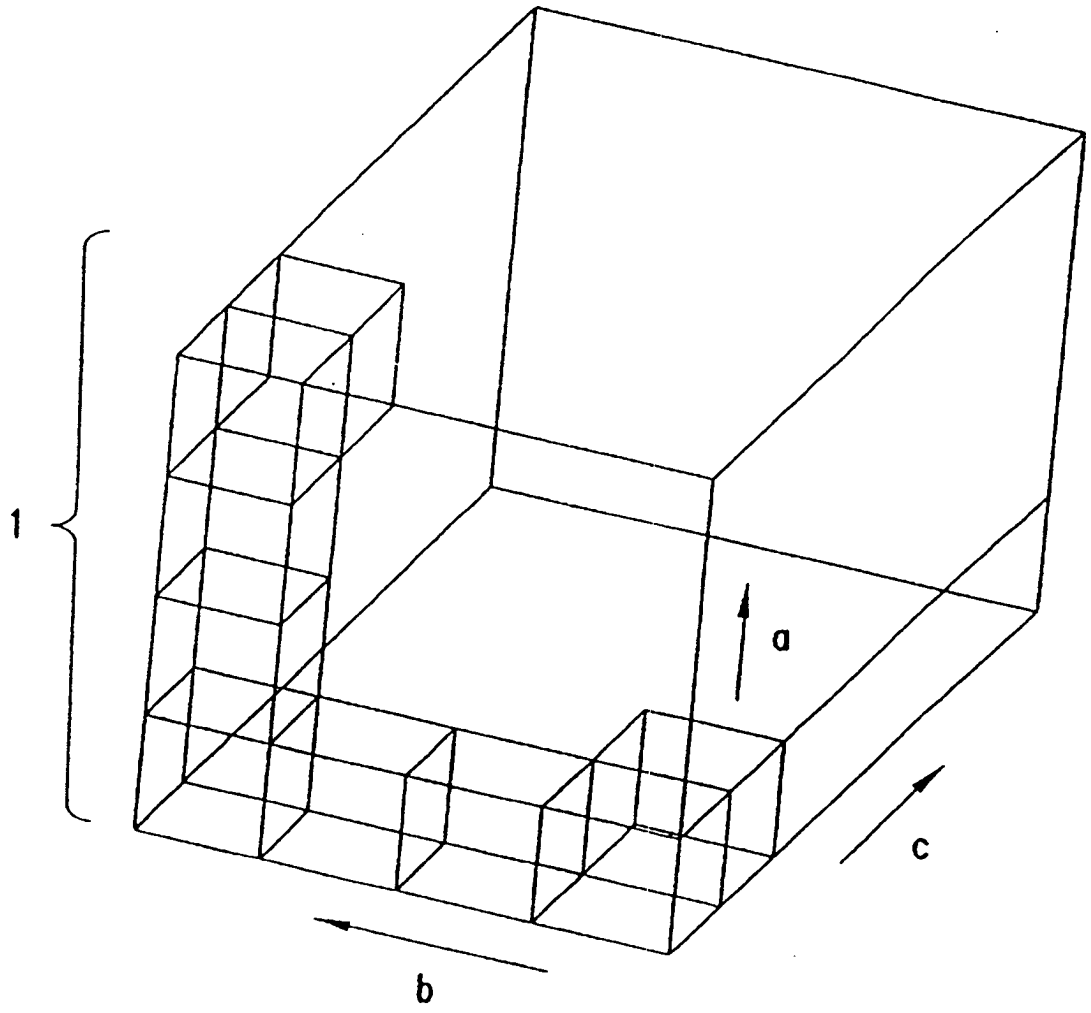


FIG. 1A

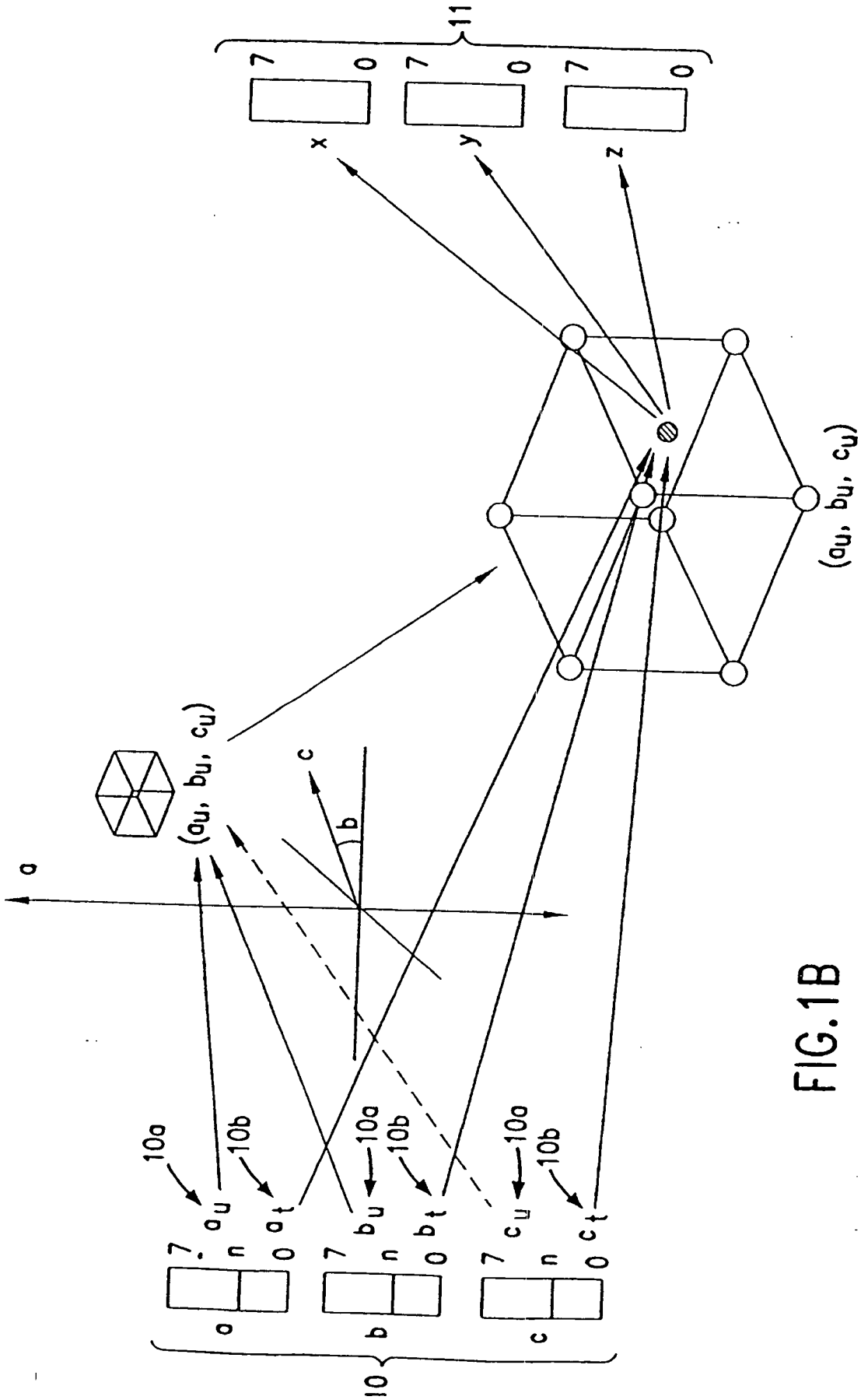


FIG.1B

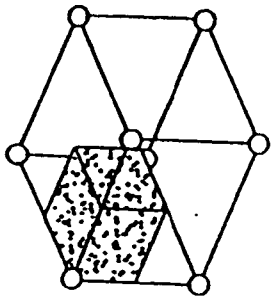


FIG.2D

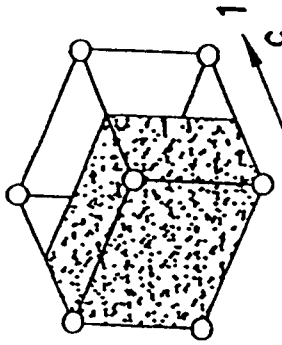


FIG.2C

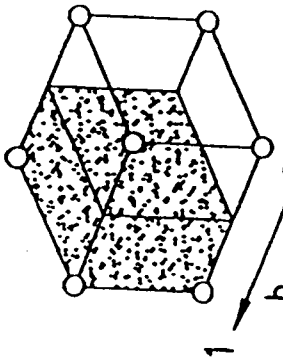


FIG.2B

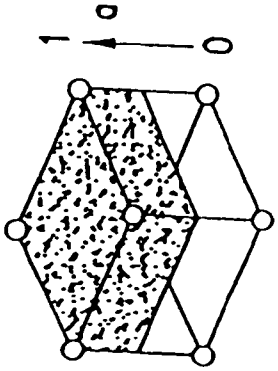


FIG.2A

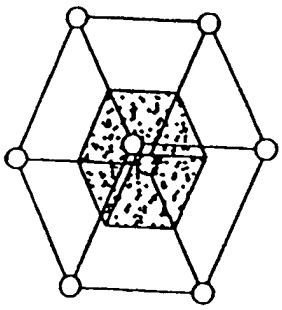


FIG.3D

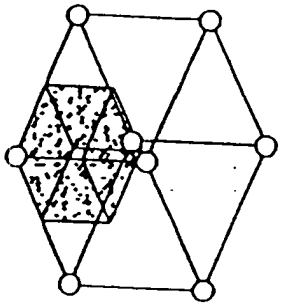


FIG.3H

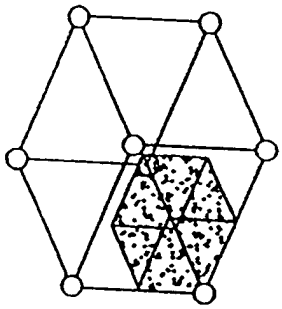


FIG.3C

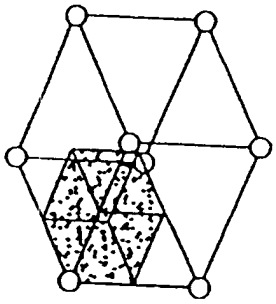


FIG.3G

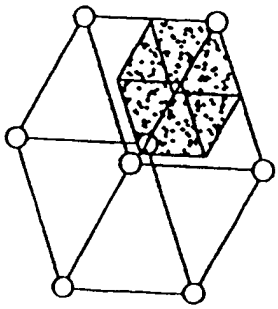


FIG.3B

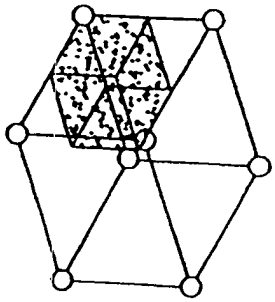


FIG.3F

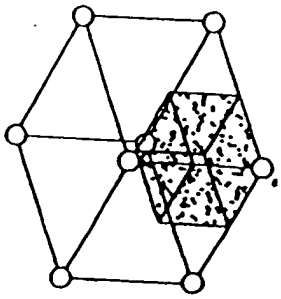


FIG.3A

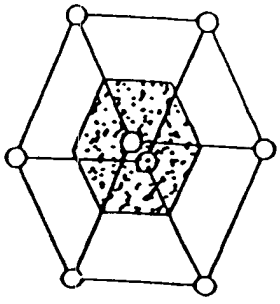


FIG.3E

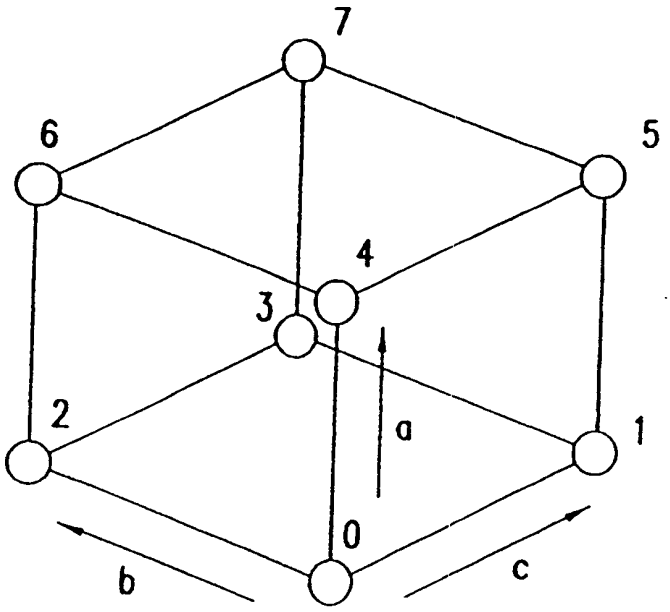


FIG.4

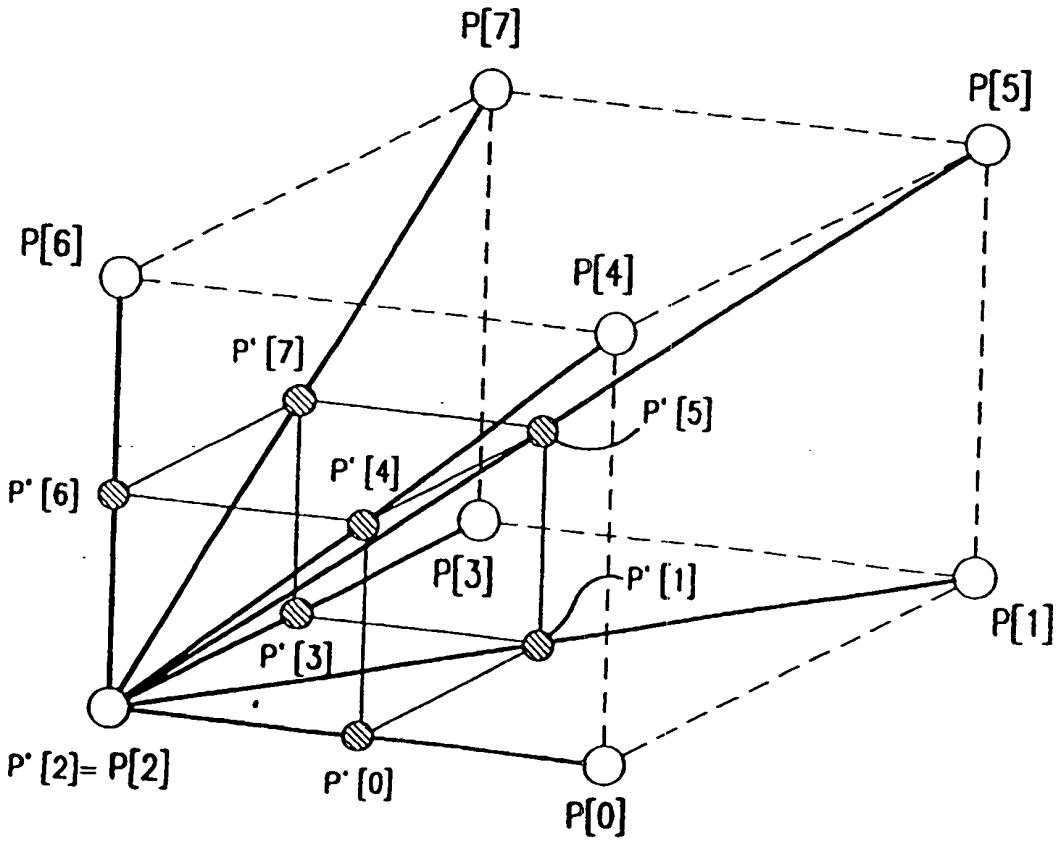


FIG.5

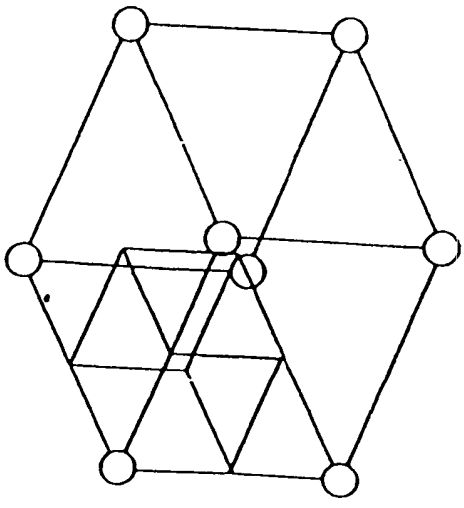


FIG.6A

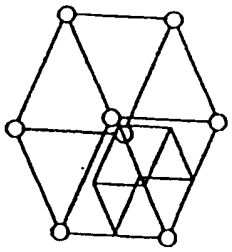


FIG.6B

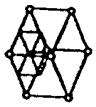


FIG.6C



FIG.6D



FIG.6E

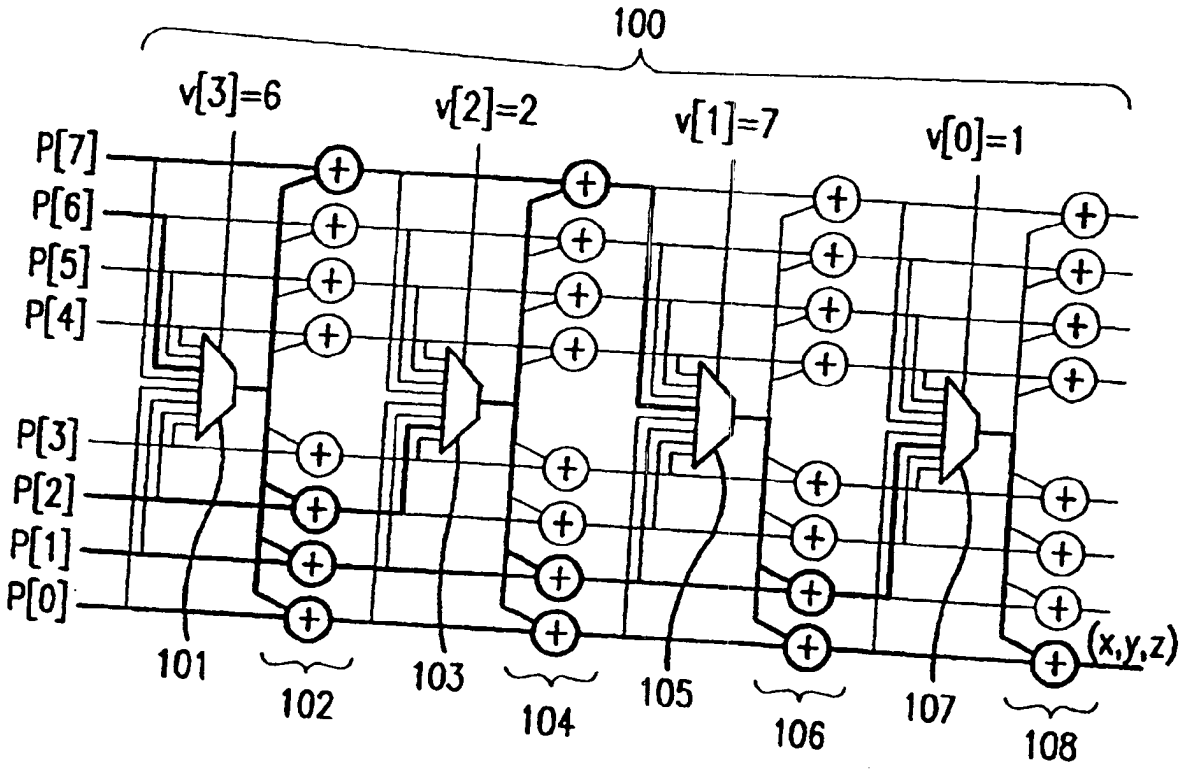

100
v[3]=6
v[2]=2
v[1]=7
v[0]=1
P[7]
P[6]
P[5]
P[4]
P[3]
P[2]
P[1]
P[0]
101
102
103
104
105
106
107
108
(x,y,z)

FIG.7

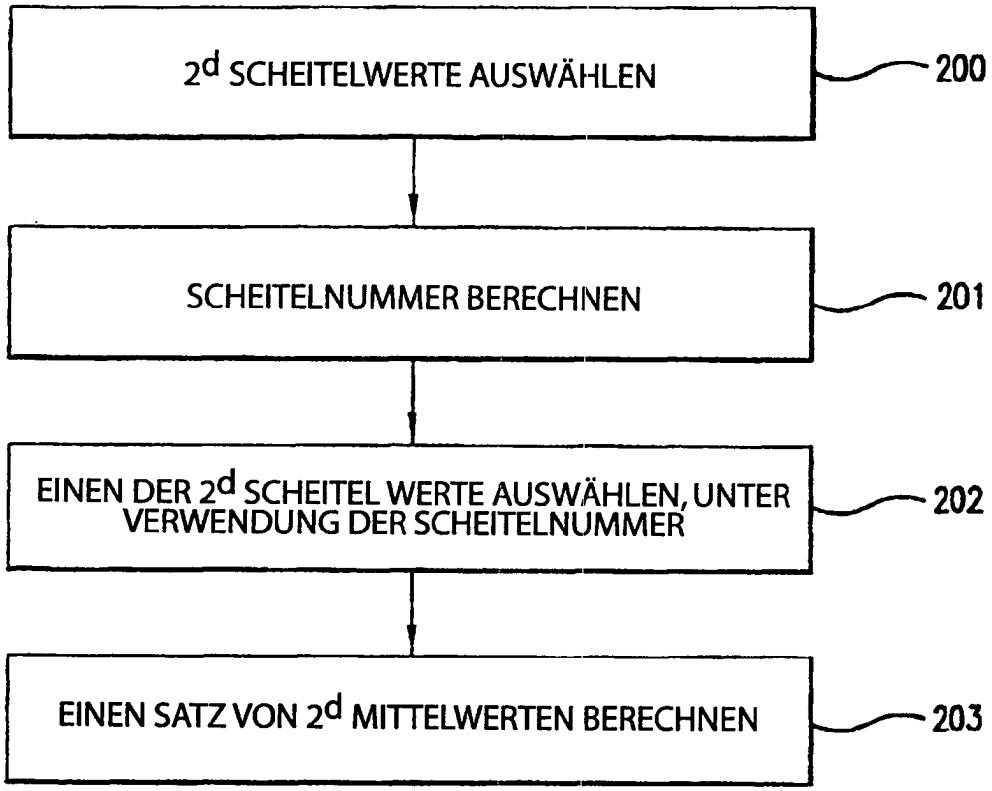

2^d SCHEITELWERTE AUSWÄHLEN
200
SCHEITELNUMMER BERECHNEN
201
EINEN DER 2^d SCHEITEL WERTE AUSWÄHLEN, UNTER VERWENDUNG DER SCHEITELNUMMER
202
EINEN SATZ VON 2^d MITTELWERTEN BERECHNEN
203

FIG.8

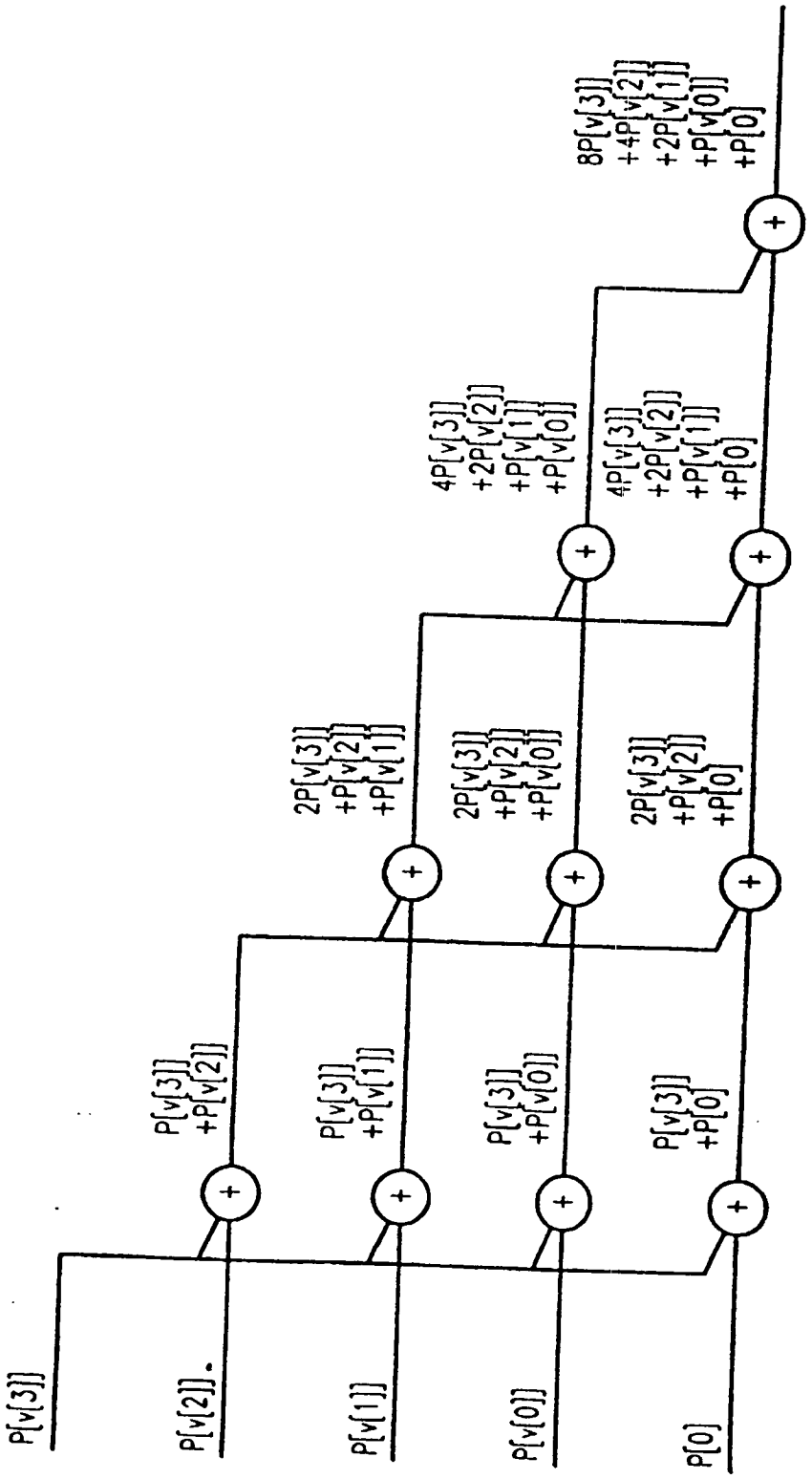

P[v[3]]
P[v[2]]
P[v[1]]
P[v[0]]
P[0]
P[v[3]] +P[v[2]]
P[v[3]] +P[v[1]]
P[v[3]] +P[v[0]]
P[v[3]] +P[0]
2P[v[3]] +P[v[2]] +P[v[1]]
2P[v[3]] +P[v[2]] +P[v[0]]
2P[v[3]] +P[v[2]] +P[0]
4P[v[3]] +2P[v[2]] +P[v[1]] +P[v[0]]
4P[v[3]] +2P[v[2]] +P[v[1]] +P[0]
8P[v[3]] +4P[v[2]] +2P[v[1]] +P[v[0]] +P[0]

FIG.9

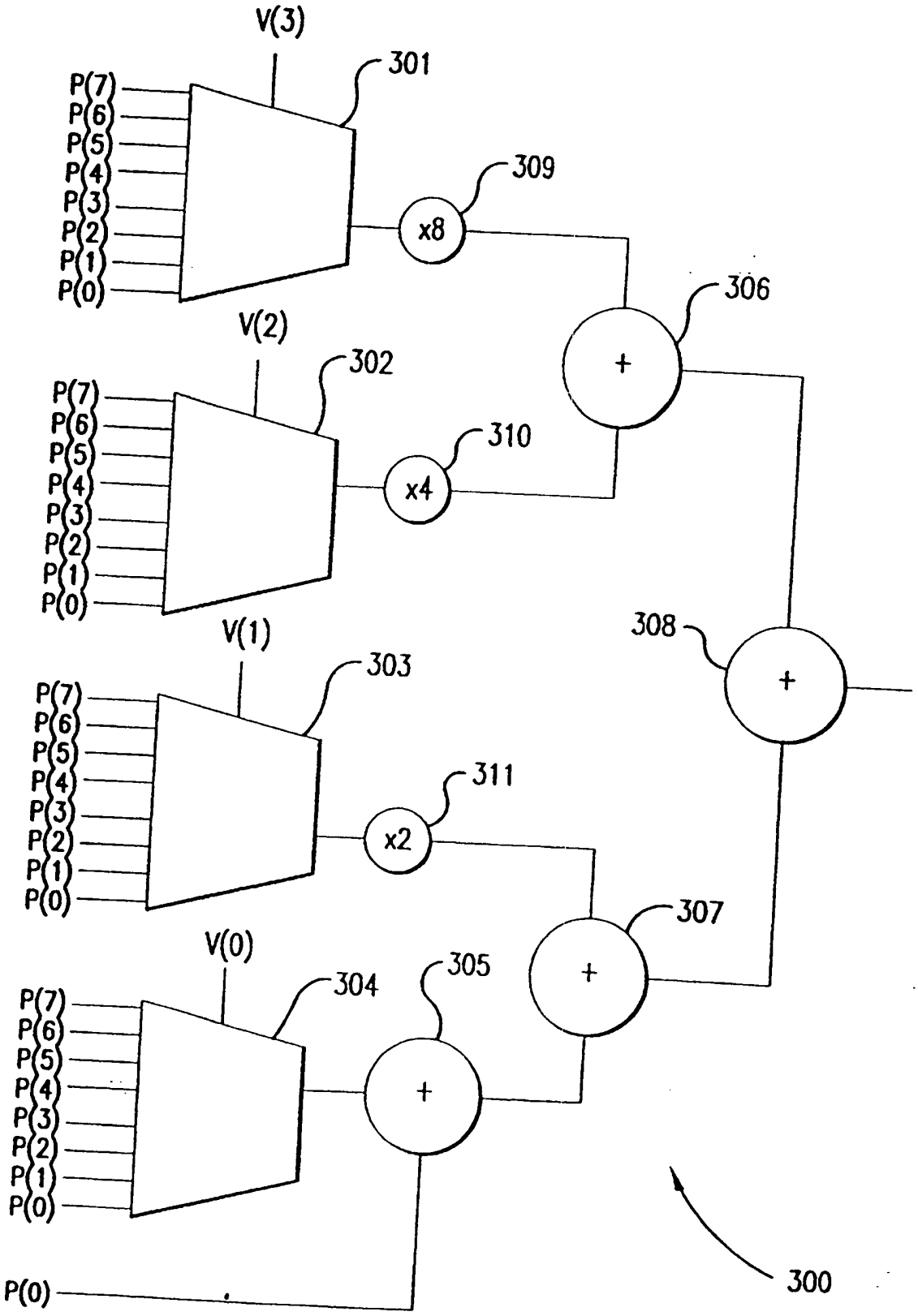

V(3)
301
P(7)
P(6)
P(5)
P(4)
P(3)
P(2)
P(1)
P(0)
309
x8
306
+
V(2)
302
310
x4
308
+
V(1)
303
311
x2
307
+
V(0)
304
305
+
P(0)
300

FIG.10

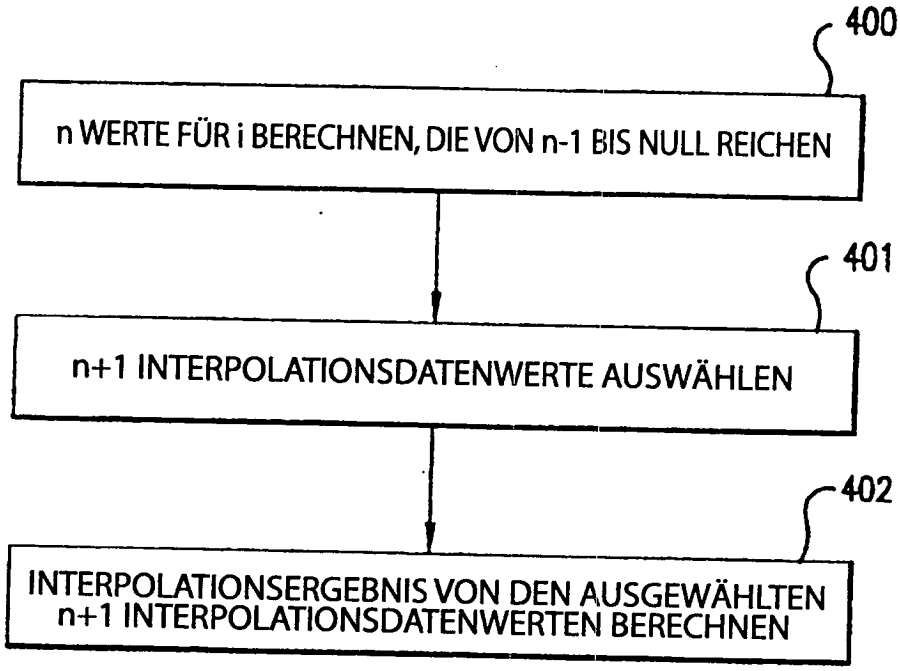

400
n WERTE FÜR i BERECHNEN, DIE VON n-1 BIS NULL REICHEN
401
n+1 INTERPOLATIONSDATENWERTE AUSWÄHLEN
402
INTERPOLATIONSERGEBNIS VON DEN AUSGEWÄHLTEN n+1 INTERPOLATIONSDATENWERTEN BERECHNEN

FIG.11

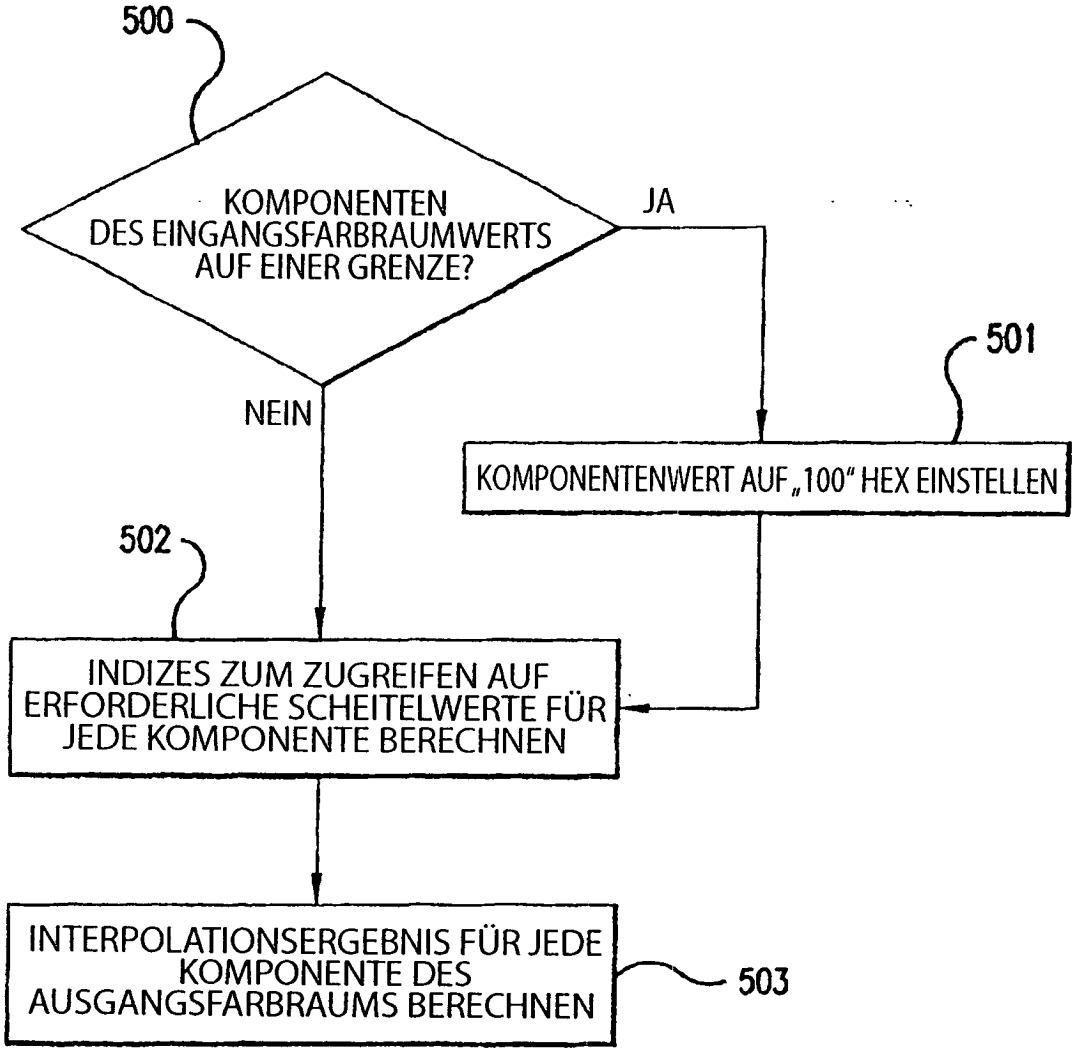

500
KOMPONENTEN DES EINGANGSFARBRAUMWERTS AUF EINER GRENZE?
JA
NEIN
501
KOMPONENTENWERT AUF „100" HEX EINSTELLEN
502
INDIZES ZUM ZUGREIFEN AUF ERFORDERLICHE SCHEITELWERTE FÜR JEDE KOMPONENTE BERECHNEN
INTERPOLATIONSERGEBNIS FÜR JEDE KOMPONENTE DES AUSGANGSFARBRAUMS BERECHNEN
503

FIG.12

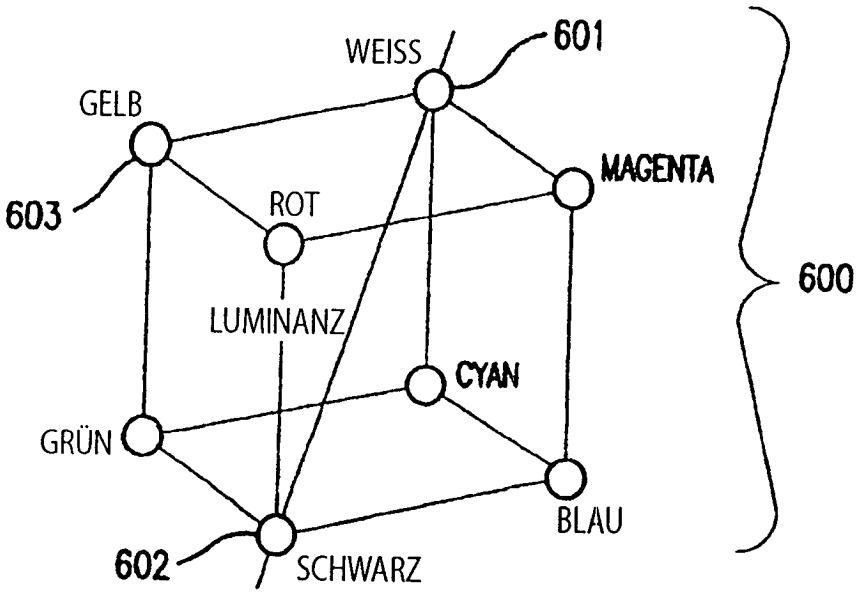

WEISS
601
GELB
MAGENTA
603
ROT
600
LUMINANZ
CYAN
GRÜN
BLAU
602
SCHWARZ

FIG.13

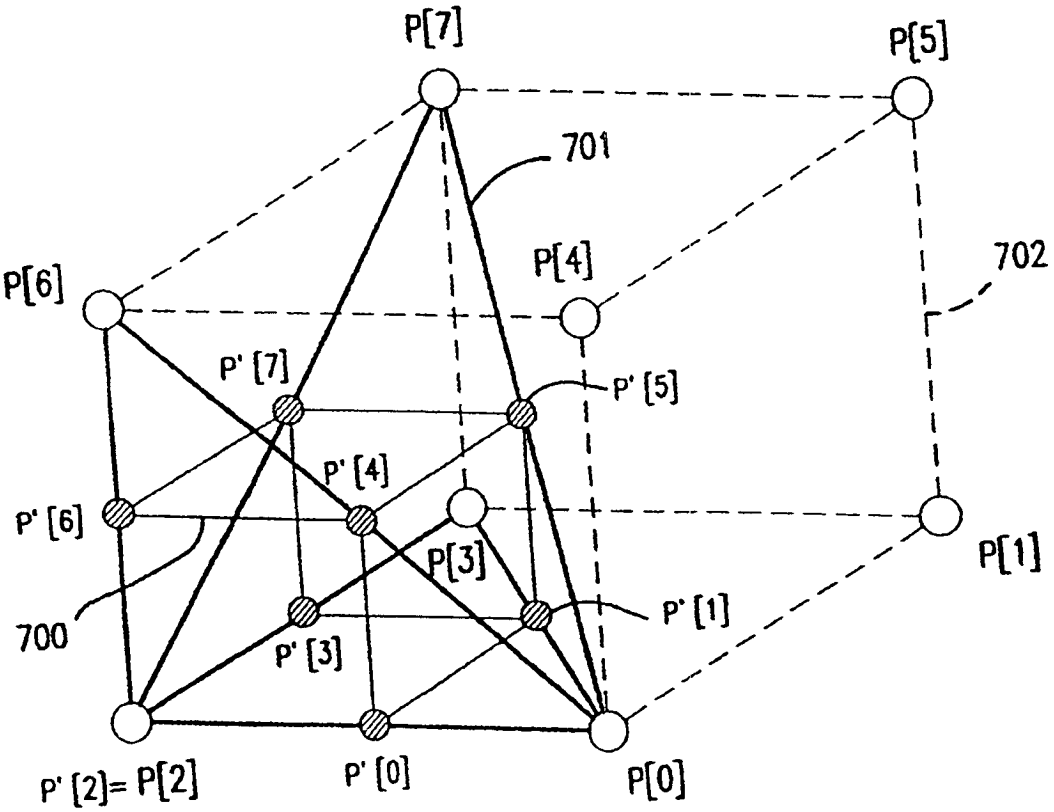


FIG.14

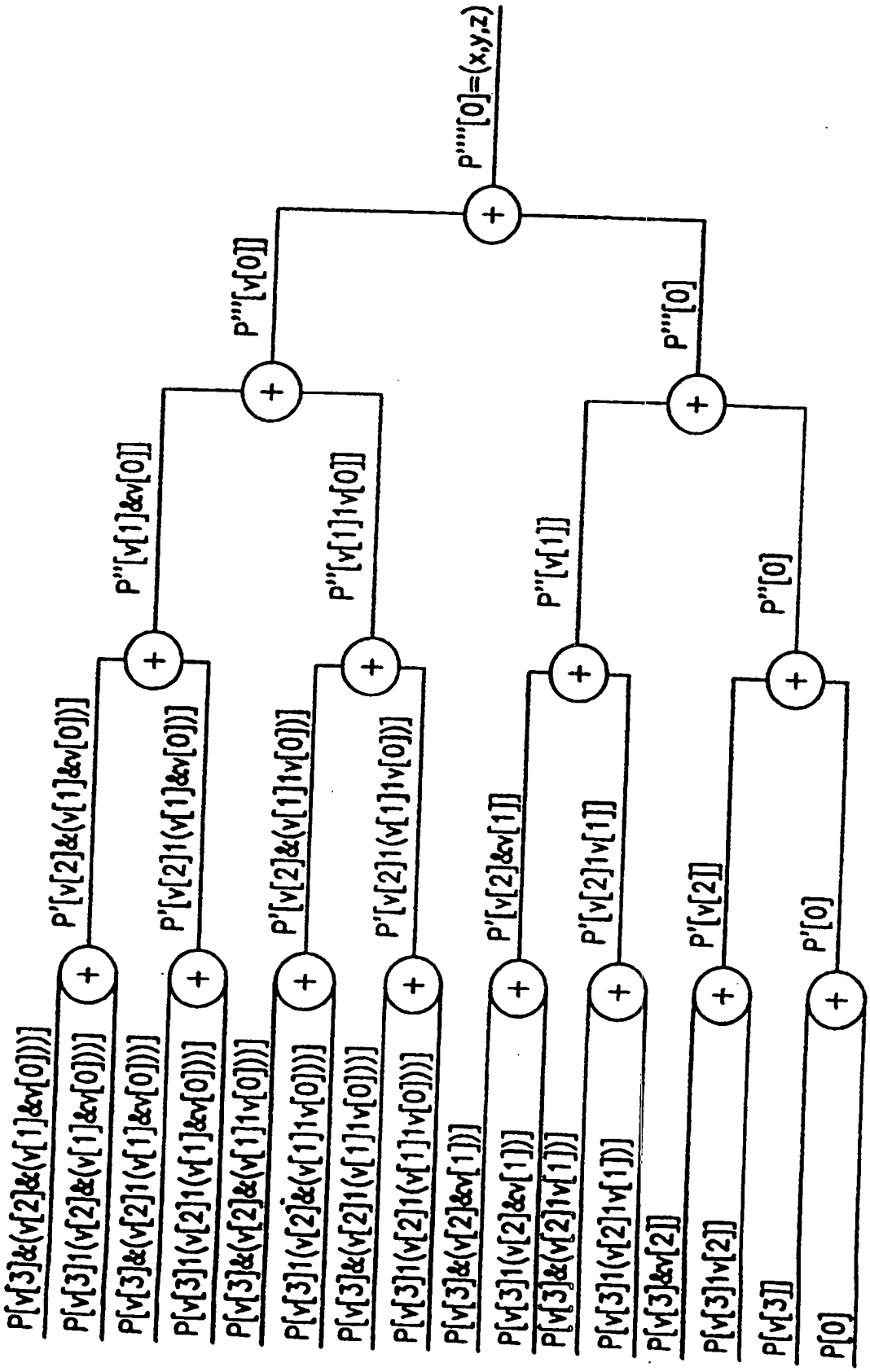

P'''[0]=(x,y,z)
P'''[v[0]]
P'''[0]
P''[v[1]&v[0]]
P''[v[1]|v[0]]
P''[v[1]]
P''[0]
P'[v[2]&(v[1]&v[0])]
P'[v[2]|(v[1]&v[0])]
P'[v[2]&(v[1]|v[0])]
P'[v[2]|(v[1]|v[0])]
P'[v[2]&v[1]]
P'[v[2]|v[1]]
P'[v[2]]
P'[0]
P[v[3]&(v[2]&(v[1]&v[0]))]
P[v[3]|(v[2]&(v[1]&v[0]))]
P[v[3]&(v[2]|(v[1]&v[0]))]
P[v[3]|(v[2]|(v[1]&v[0]))]
P[v[3]&(v[2]&(v[1]|v[0]))]
P[v[3]|(v[2]&(v[1]|v[0]))]
P[v[3]&(v[2]|(v[1]|v[0]))]
P[v[3]|(v[2]|(v[1]|v[0]))]
P[v[3]&(v[2]&v[1])]
P[v[3]|(v[2]&v[1])]
P[v[3]&(v[2]|v[1])]
P[v[3]|(v[2]|v[1])]
P[v[3]&v[2]]
P[v[3]|v[2]]
P[v[3]]
P[0]

FIG. 15

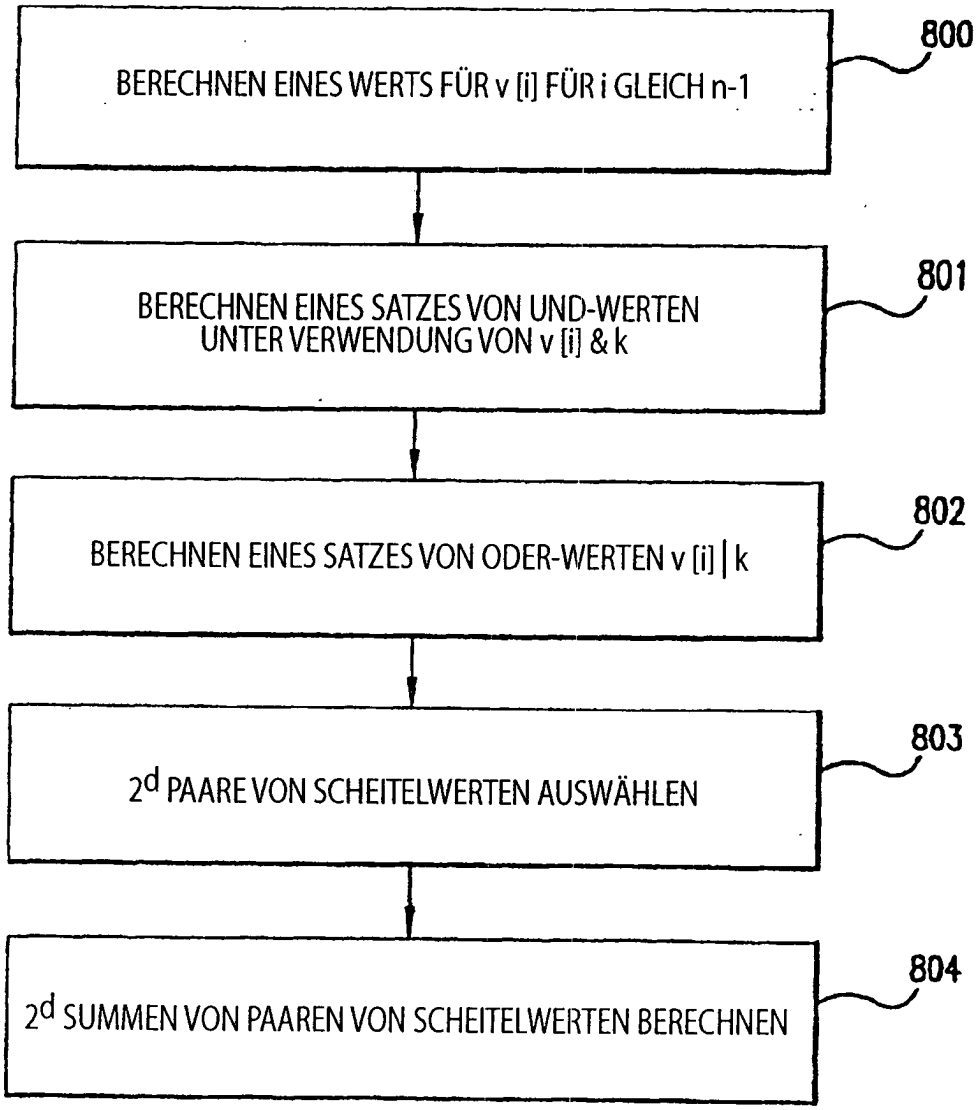

BERECHNEN EINES WERTS FÜR v [i] FÜR i GLEICH n-1
800
BERECHNEN EINES SATZES VON UND-WERTEN UNTER VERWENDUNG VON v [i] & k
801
BERECHNEN EINES SATZES VON ODER-WERTEN v [i] | k
802
2d PAARE VON SCHEITELWERTEN AUSWÄHLEN
803
2d SUMMEN VON PAAREN VON SCHEITELWERTEN BERECHNEN
804

FIG.16

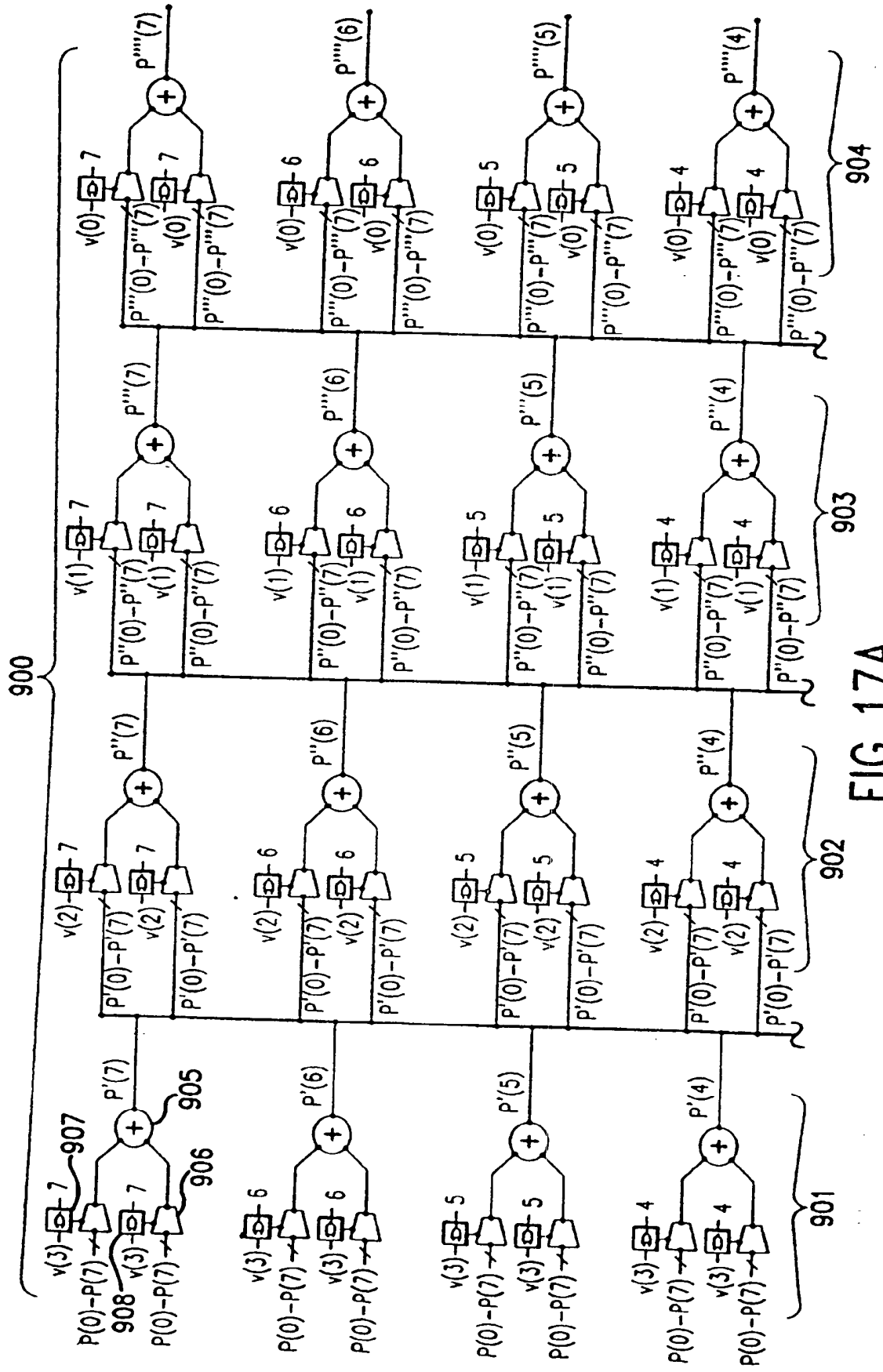

900
901
902
903
904
905
906
907
908
P(0)-P(7)
P'(0)-P'(7)
P''(0)-P''(7)
P'''(0)-P'''(7)
P''''(0)-P''''(7)
P'(7)
P'(6)
P'(5)
P'(4)
P''(7)
P''(6)
P''(5)
P''(4)
P'''(7)
P'''(6)
P'''(5)
P'''(4)
P''''(7)
P''''(6)
P''''(5)
P''''(4)
v(0)
v(1)
v(2)
v(3)

FIG. 17A

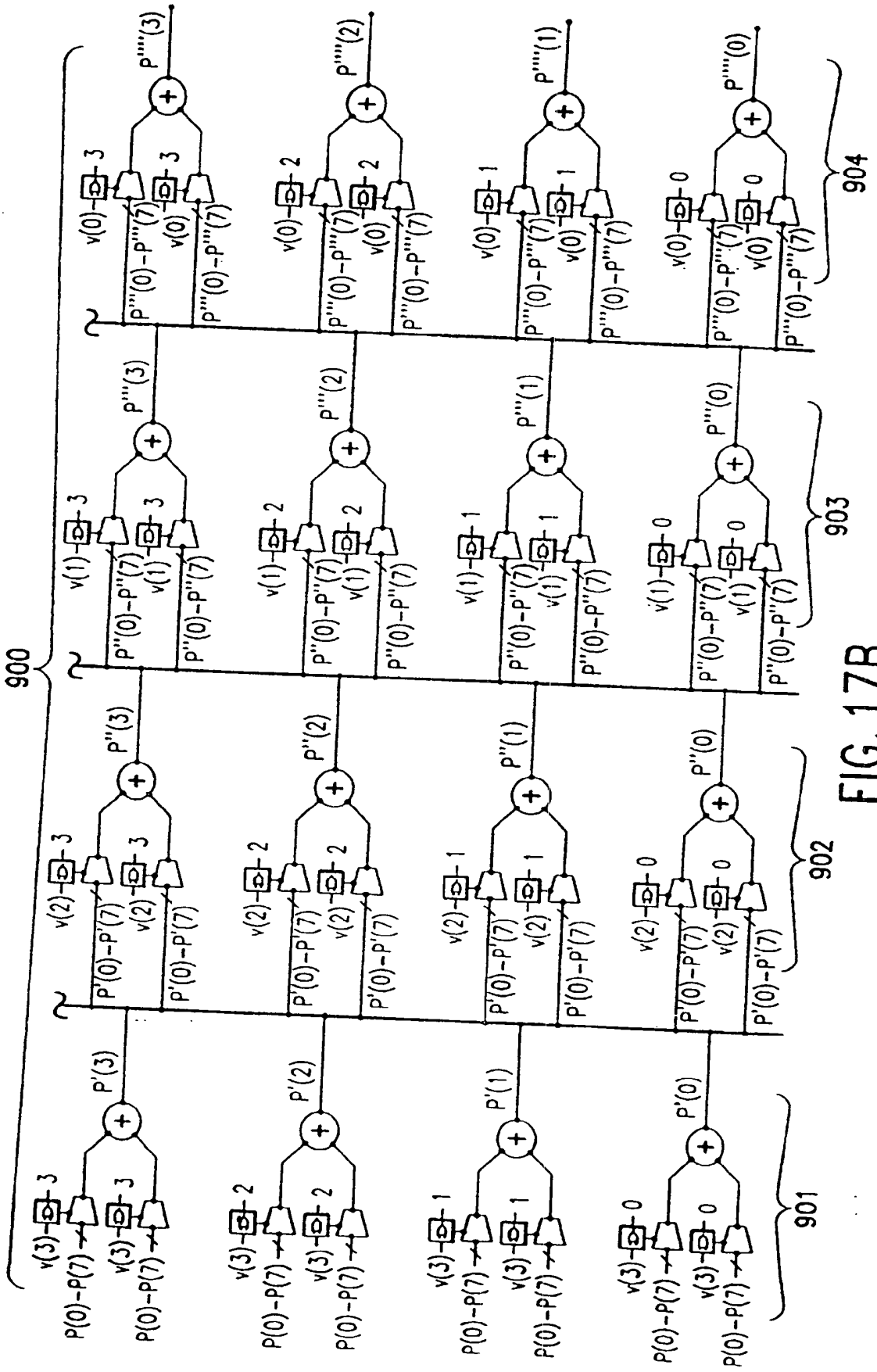

900
901
902
903
904
P(0)-P(7)
P'(0)-P'(7)
P''(0)-P''(7)
P'''(0)-P'''(7)
P'(3)
P'(2)
P'(1)
P'(0)
P''(3)
P''(2)
P''(1)
P''(0)
P'''(3)
P'''(2)
P'''(1)
P'''(0)
P''''(3)
P''''(2)
P''''(1)
P''''(0)
v(3)
v(2)
v(1)
v(0)

FIG. 17B

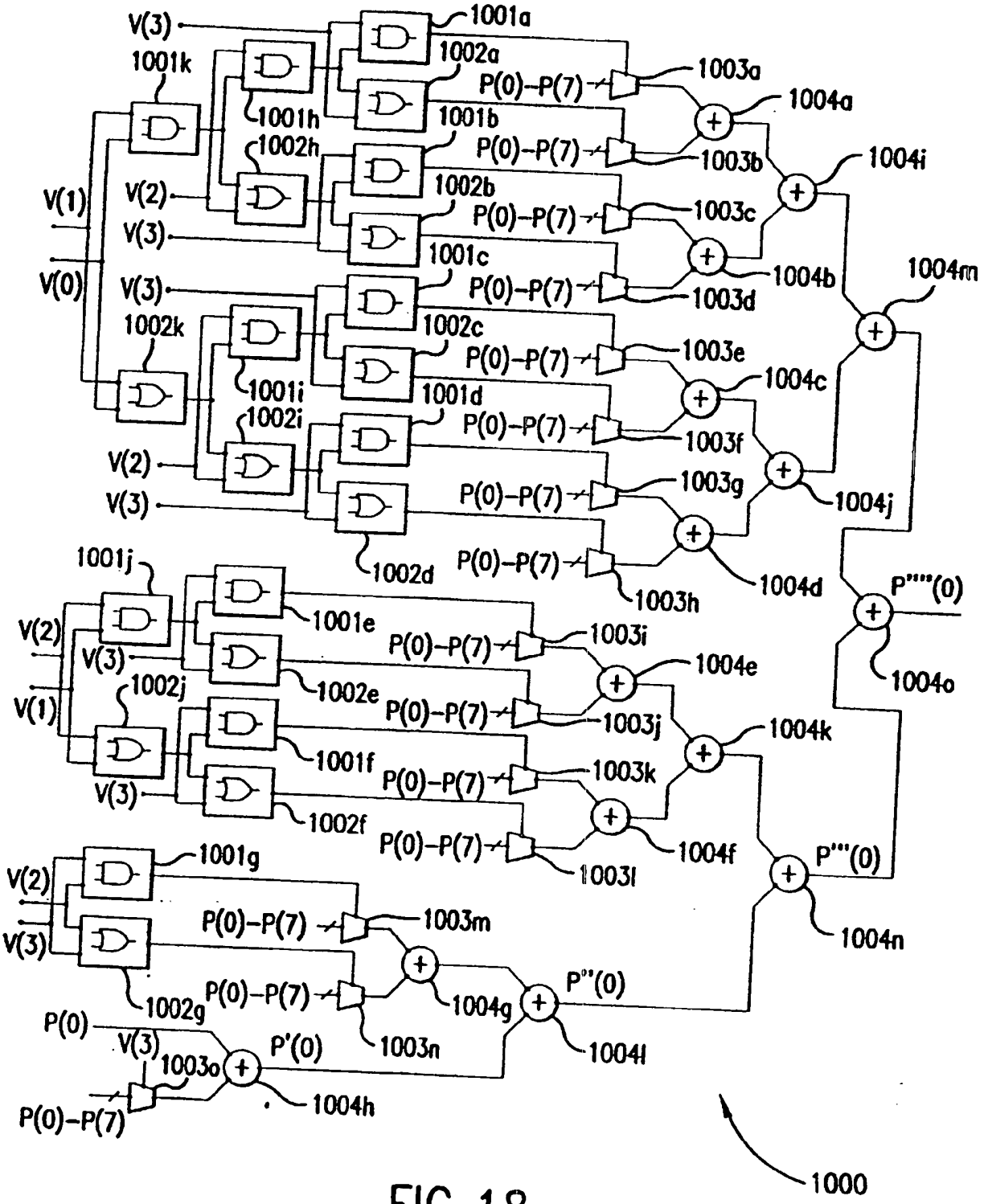

V(3)
1001a
1001k
1002a
P(0)–P(7)
1003a
1004a
1001h
1001b
1002h
P(0)–P(7)
1003b
1004i
V(1)
V(2)
1002b
P(0)–P(7)
1003c
V(3)
1001c
1004b
1004m
V(0)
V(3)
P(0)–P(7)
1003d
1002k
1002c
P(0)–P(7)
1003e
1001i
1001d
1004c
1002i
P(0)–P(7)
1003f
V(2)
1003g
1004j
V(3)
P(0)–P(7)
1002d
P(0)–P(7)
1003h
1004d
P''''(0)
1001j
1001e
P(0)–P(7)
1003i
V(2)
1004e
V(3)
1002e
P(0)–P(7)
1003j
1004o
1002j
V(1)
1004k
1001f
P(0)–P(7)
1003k
V(3)
1002f
1004f
P(0)–P(7)
1003l
P'''(0)
1001g
V(2)
P(0)–P(7)
1003m
V(3)
1004n
P(0)–P(7)
1004g
P''(0)
1002g
1003n
1004l
P(0)
V(3)
1003o
P'(0)
1004h
P(0)–P(7)
1000

FIG. 18

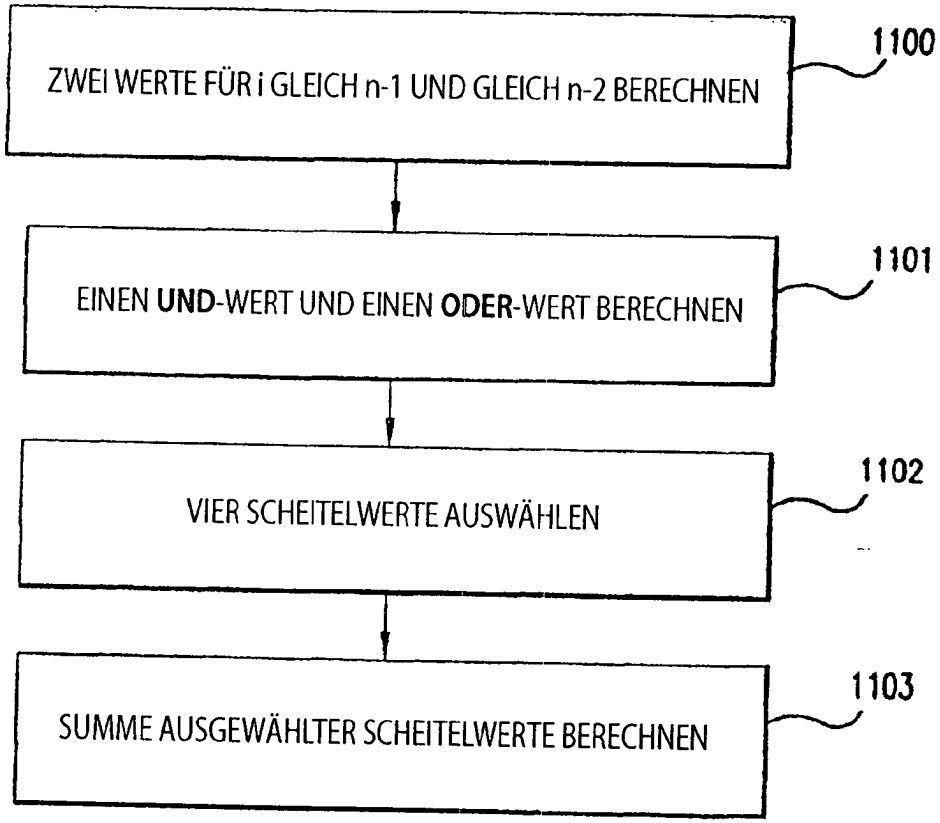

ZWEI WERTE FÜR i GLEICH n-1 UND GLEICH n-2 BERECHNEN
1100
EINEN **UND**-WERT UND EINEN **ODER**-WERT BERECHNEN
1101
VIER SCHEITELWERTE AUSWÄHLEN
1102
SUMME AUSGEWÄHLTER SCHEITELWERTE BERECHNEN
1103

FIG.19

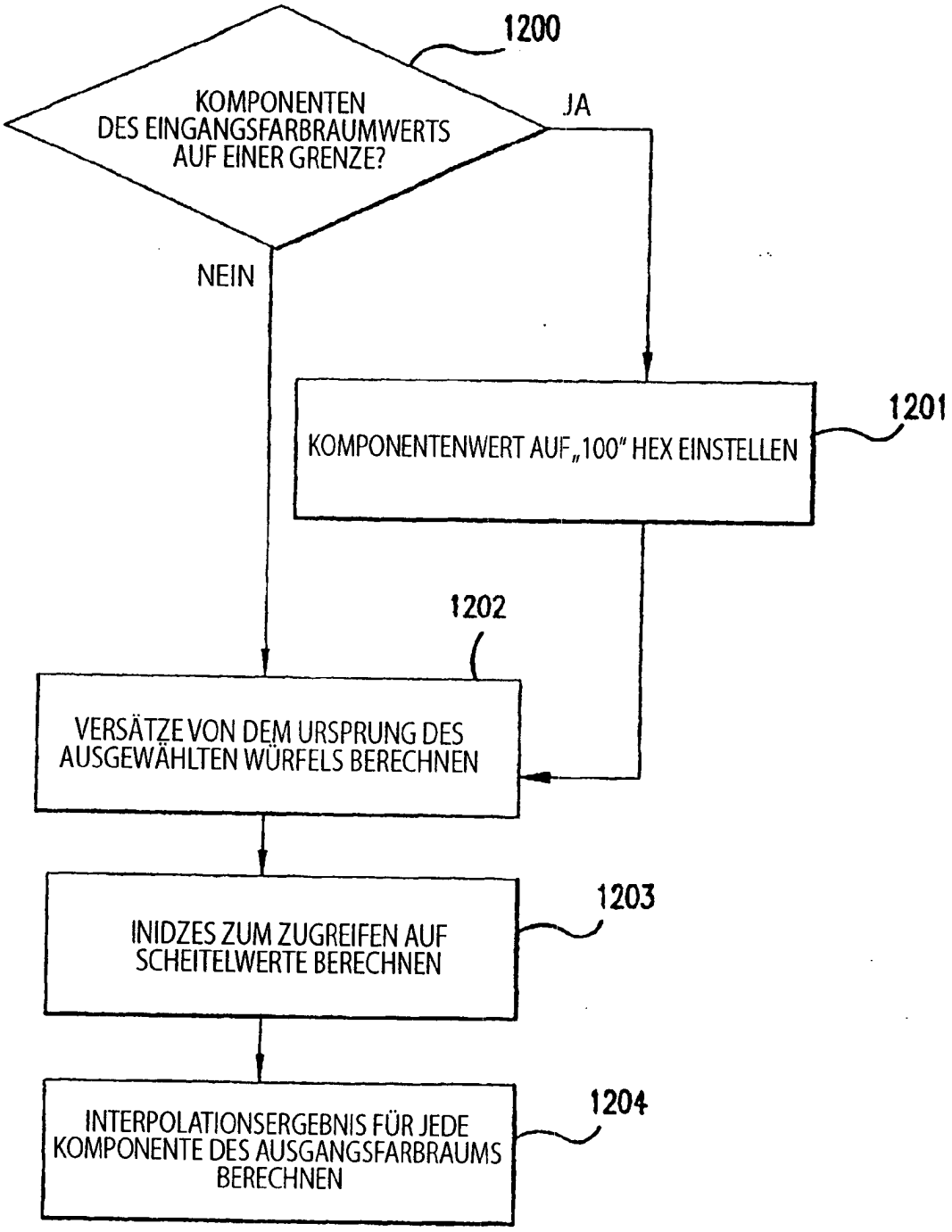

1200
KOMPONENTEN DES EINGANGSFARBRAUMWERTS AUF EINER GRENZE?
JA
NEIN
KOMPONENTENWERT AUF „100" HEX EINSTELLEN
1201
1202
VERSÄTZE VON DEM URSPRUNG DES AUSGEWÄHLTEN WÜRFELS BERECHNEN
INIDZES ZUM ZUGREIFEN AUF SCHEITELWERTE BERECHNEN
1203
INTERPOLATIONSERGEBNIS FÜR JEDE KOMPONENTE DES AUSGANGSFARBRAUMS BERECHNEN
1204

FIG.20

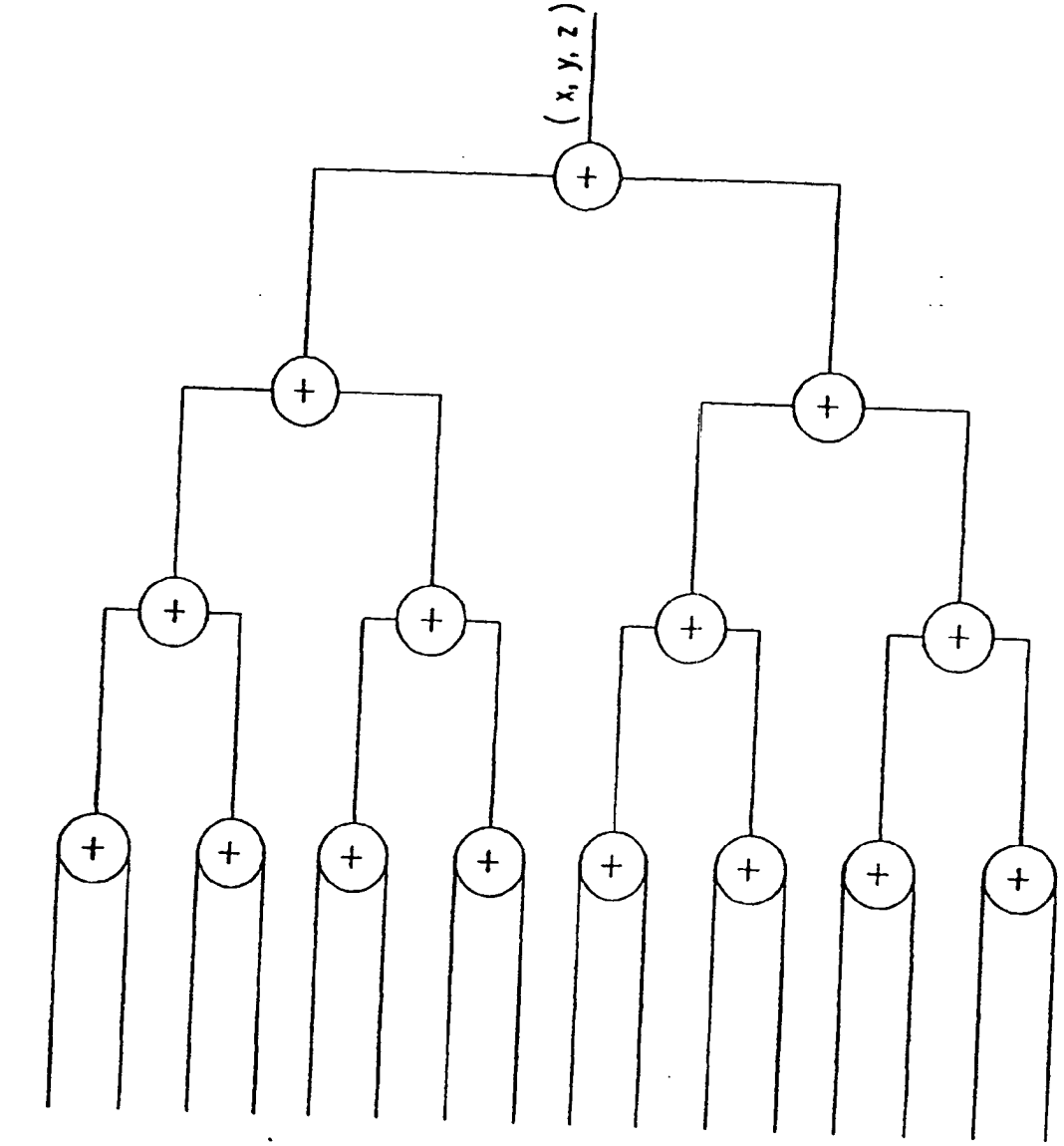
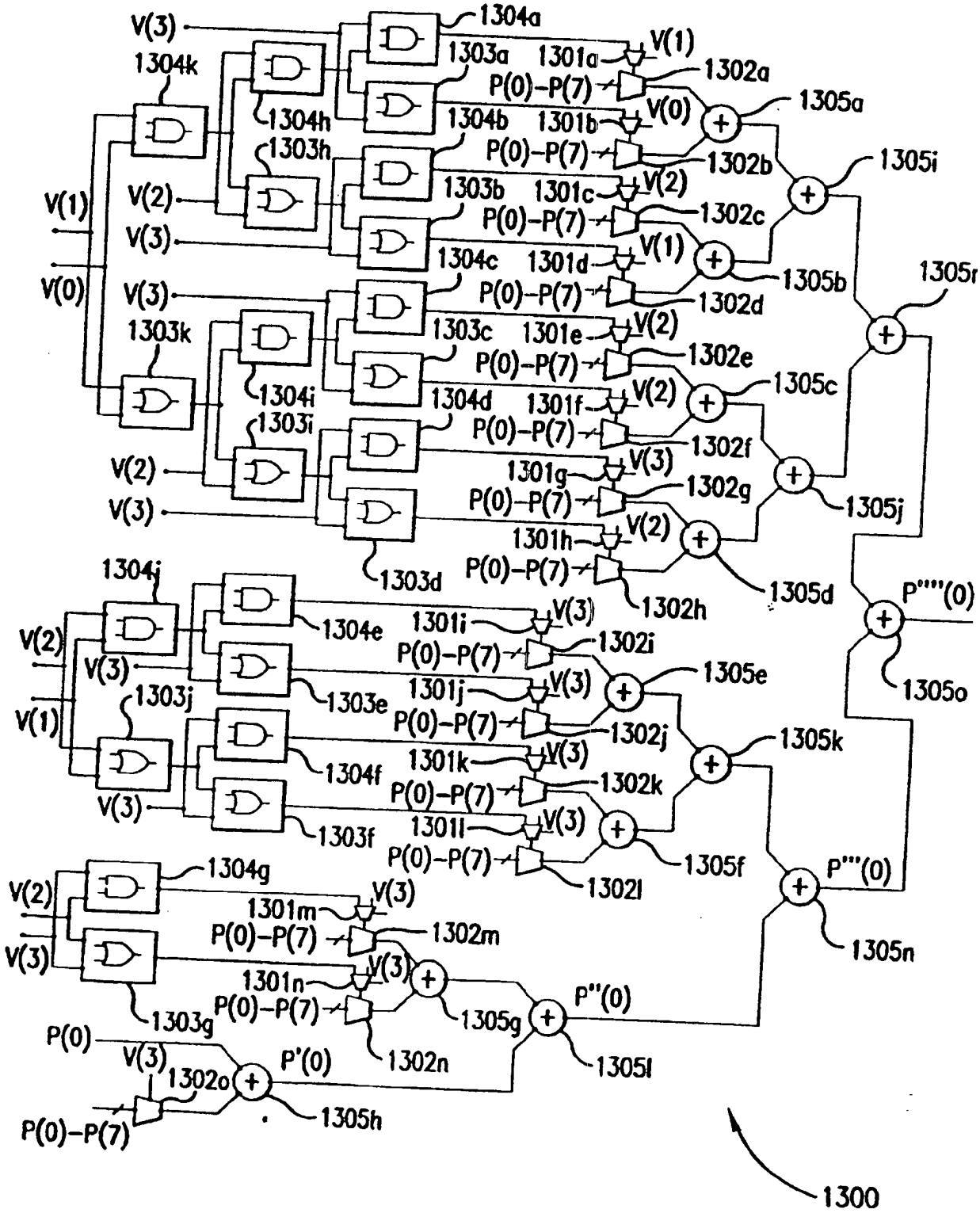

(x, y, z)
P[v[0]] P[v[3] & (v[2] & (v[1] & v[0]))]
P[v[1]] P[v[3] | (v[2] & (v[1] & v[0]))]
P[v[1]] P[v[3] & (v[2] | (v[1] & v[0]))]
P[v[2]] P[v[3] | (v[2] | (v[1] & v[0]))]
P[v[2]] P[v[3] & (v[2] & (v[1] | v[0]))]
P[v[2]] P[v[3] | (v[2] & (v[1] | v[0]))]
P[v[2]] P[v[3] & (v[2] | (v[1] | v[0]))]
P[v[3]] P[v[3] | (v[2] | (v[1] | v[0]))]
P[v[3]] P[v[3] & (v[2] & v[1])]
P[v[3]] P[v[3] | (v[2] & v[1])]
P[v[3]] P[v[3] & (v[2] | v[1])]
P[v[3]] P[v[3] | (v[2] | v[1])]
P[v[3]] P[v[3] & v[2]]
P[v[3]] P[v[3] | v[2]]
P[v[3]] P[v[3]]
P[0] P[0]

FIG.21


V(3)
1304a
1303a
1301a
V(1)
1302a
P(0)-P(7)
1305a
1304k
1304h
1304b
1301b
V(0)
1303h
1305i
V(1)
V(2)
1303b
1301c
V(2)
1302c
V(3)
1304c
1301d
V(1)
1305b
1305r
V(0)
V(3)
1303k
1303c
1301e
V(2)
1302e
1304i
1304d
1301f
V(2)
1305c
1303i
1302f
V(2)
1301g
V(3)
1302g
V(3)
1305j
1301h
V(2)
1303d
1302h
1305d
P''''(0)
1304j
1304e
1301i
V(3)
1302i
V(2)
1305e
1303e
1301j
V(3)
1303j
1302j
V(1)
1305k
1304f
1301k
V(3)
1302k
V(3)
1303f
1301l
V(3)
1305f
P'''(0)
1304g
V(2)
1301m
V(3)
1302m
1305n
V(3)
1301n
V(3)
P''(0)
1303g
1305g
1305l
P(0)
V(3)
1302o
P'(0)
1305h
1300
FIG. 22

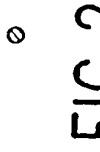


FIG.23E



FIG.23D

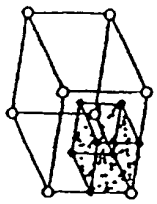


FIG.23C

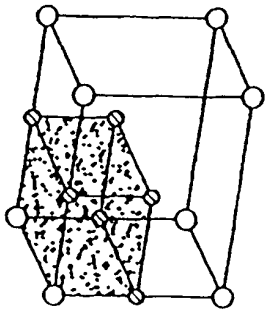


FIG.23B

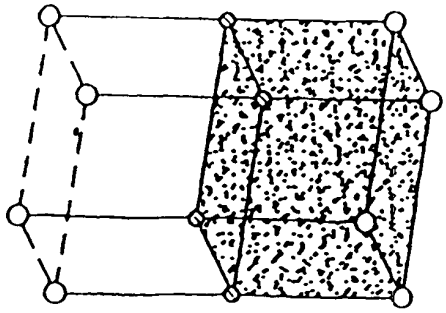


FIG.23A

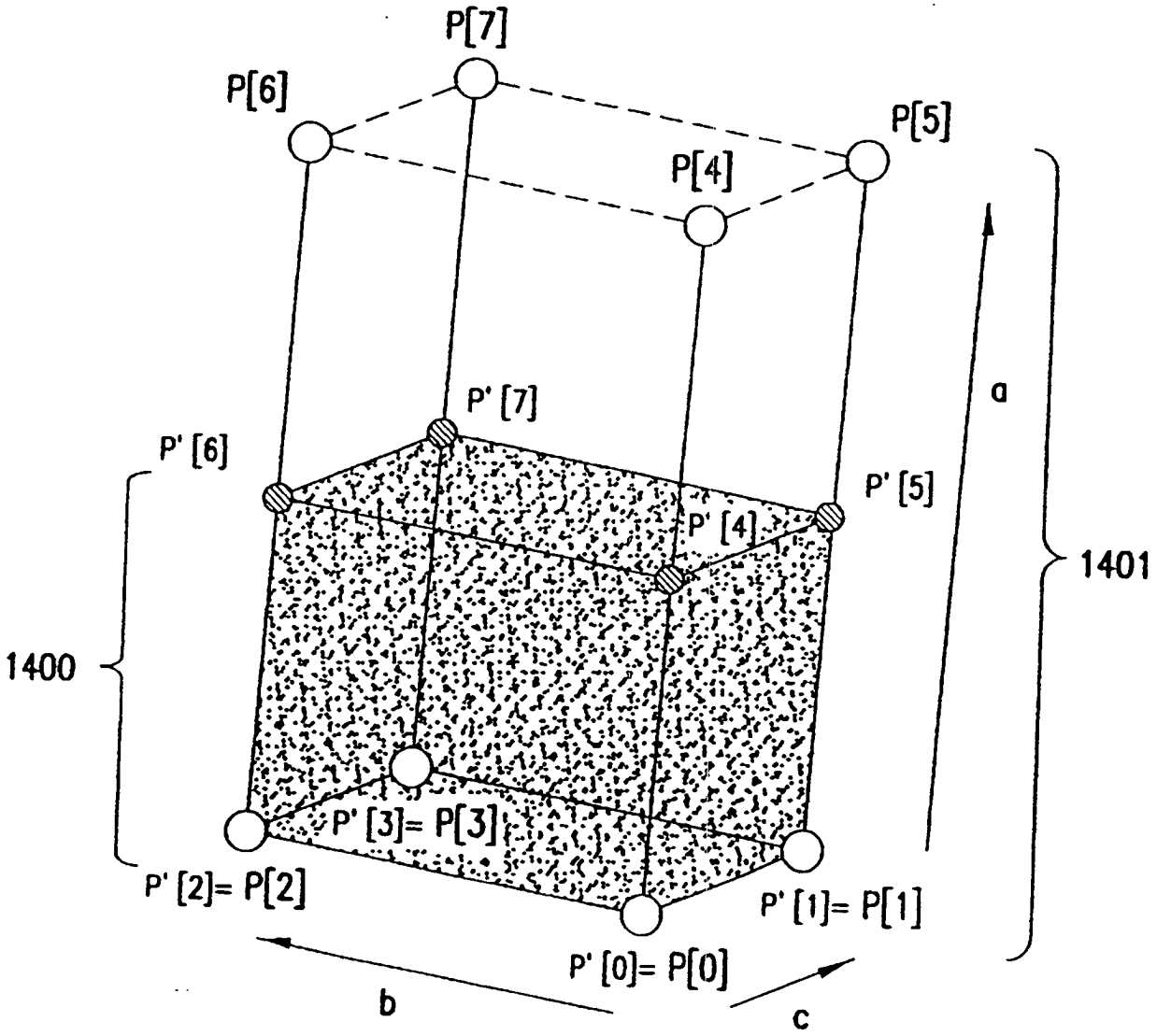


FIG.24

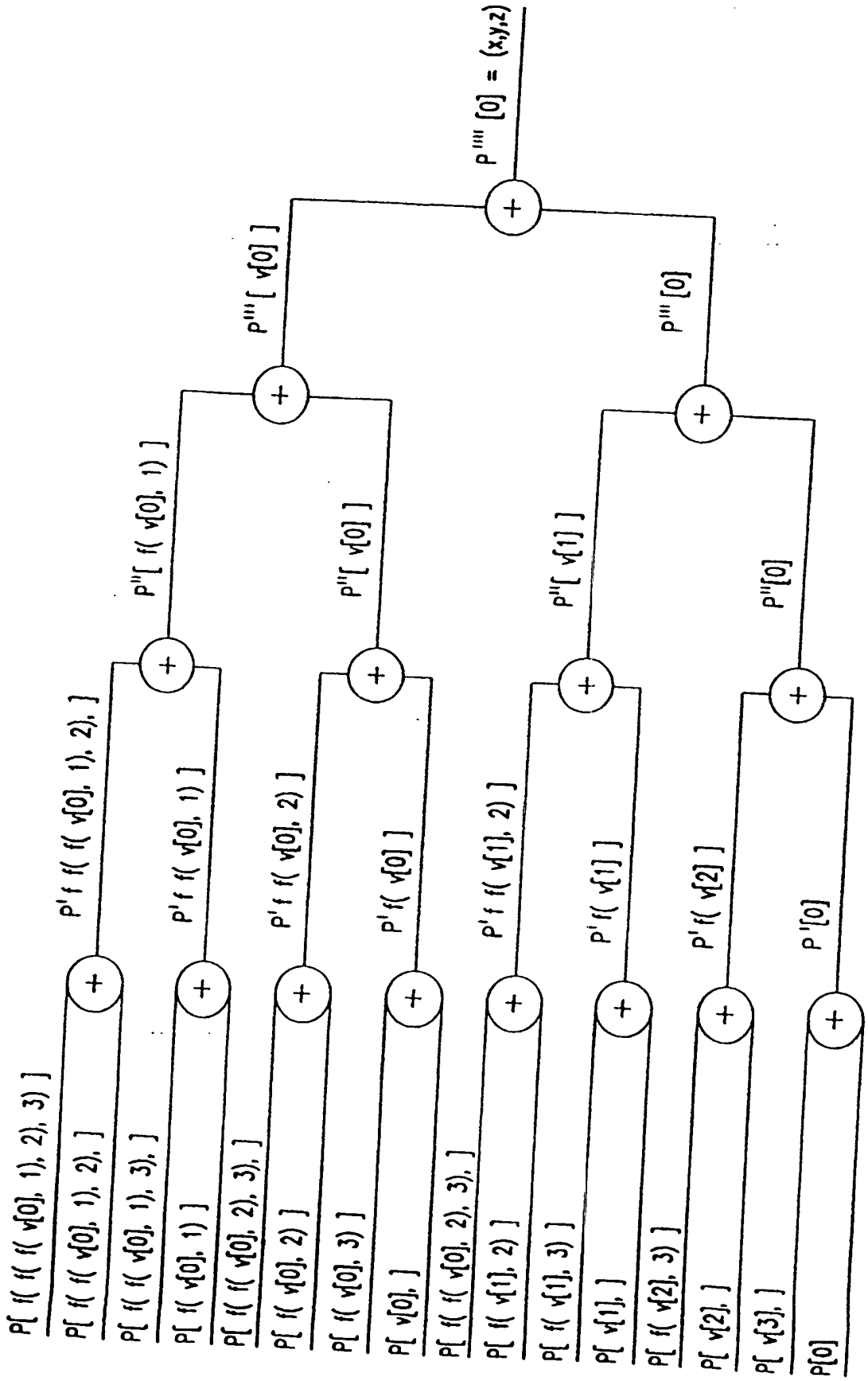

P''''[0] = (x,y,z)
P'''[v[0]]
P'''[0]
P''[f(v[0], 1)]
P''[v[0]]
P''[v[1]]
P''[0]
P'f f(f(v[0], 1), 2),]
P'f f(f(v[0], 2), 3),]
P'f f(v[0], 2)]
P'f(v[0]]
P'f f(v[1], 2)]
P'f(v[1]]
P'f(v[2]]
P'[0]
P[f(f(f(v[0], 1), 2), 3)]
P[f(f(v[0], 1), 2),]
P[f(f(v[0], 1), 3),]
P[f(v[0], 1)]
P[f(f(v[0], 2), 3),]
P[f(v[0], 2)]
P[f(v[0], 3)]
P[v[0],]
P[f(f(v[1], 2), 3),]
P[f(v[1], 2)]
P[f(v[1], 3)]
P[v[1],]
P[f(v[2], 3)]
P[v[2],]
P[v[3],]
P[0]

FIG.25

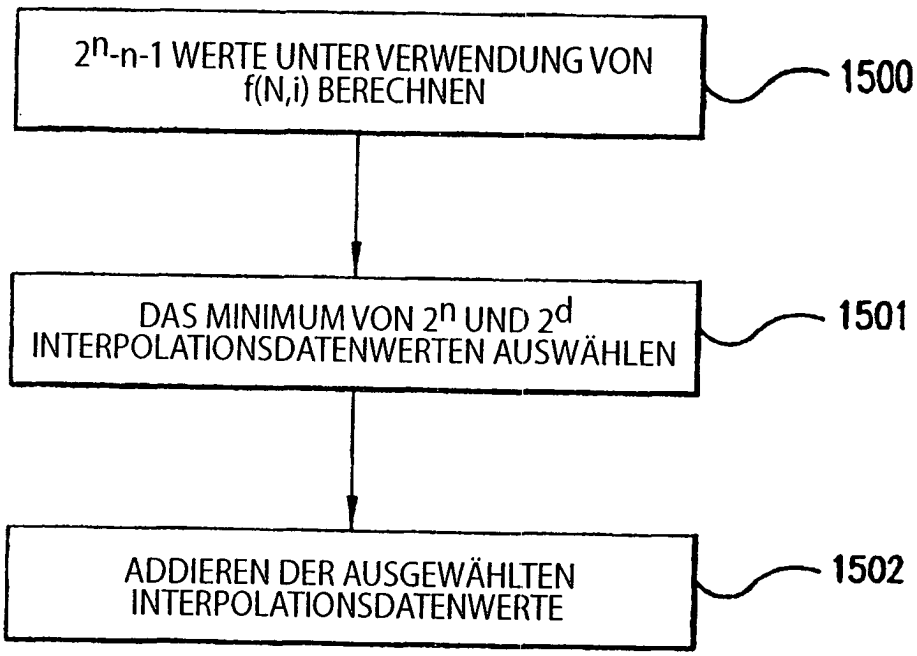

2^n-n-1 WERTE UNTER VERWENDUNG VON f(N,i) BERECHNEN
1500
DAS MINIMUM VON 2^n UND 2^d INTERPOLATIONSDATENWERTEN AUSWÄHLEN
1501
ADDIEREN DER AUSGEWÄHLTEN INTERPOLATIONSDATENWERTE
1502

FIG. 26

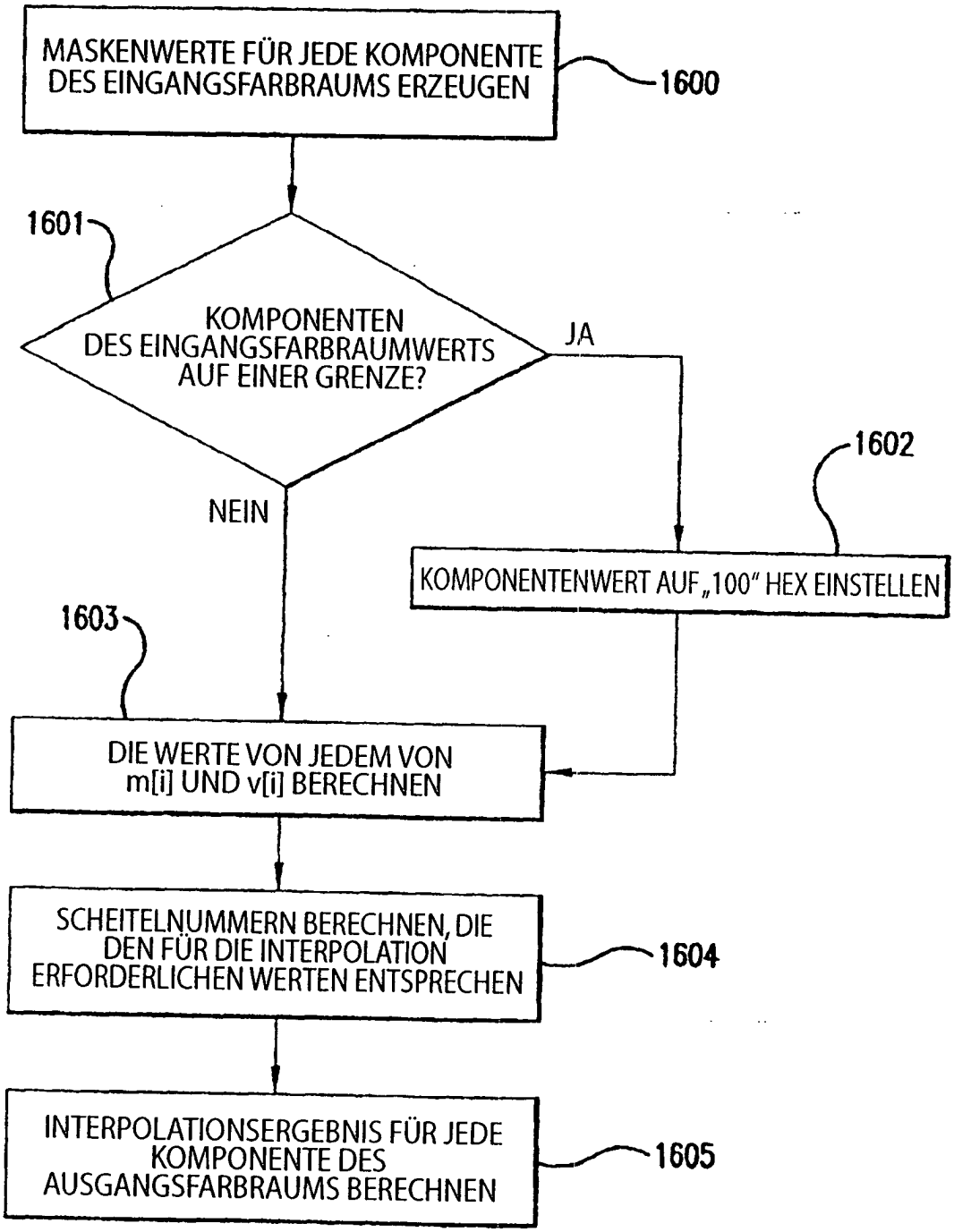

MASKENWERTE FÜR JEDE KOMPONENTE DES EINGANGSFARBRAUMS ERZEUGEN
1600
1601
KOMPONENTEN DES EINGANGSFARBRAUMWERTS AUF EINER GRENZE?
JA
NEIN
1602
KOMPONENTENWERT AUF „100" HEX EINSTELLEN
1603
DIE WERTE VON JEDEM VON m[i] UND v[i] BERECHNEN
SCHEITELNUMMERN BERECHNEN, DIE DEN FÜR DIE INTERPOLATION ERFORDERLICHEN WERTEN ENTSPRECHEN
1604
INTERPOLATIONSERGEBNIS FÜR JEDE KOMPONENTE DES AUSGANGSFARBRAUMS BERECHNEN
1605

FIG. 27

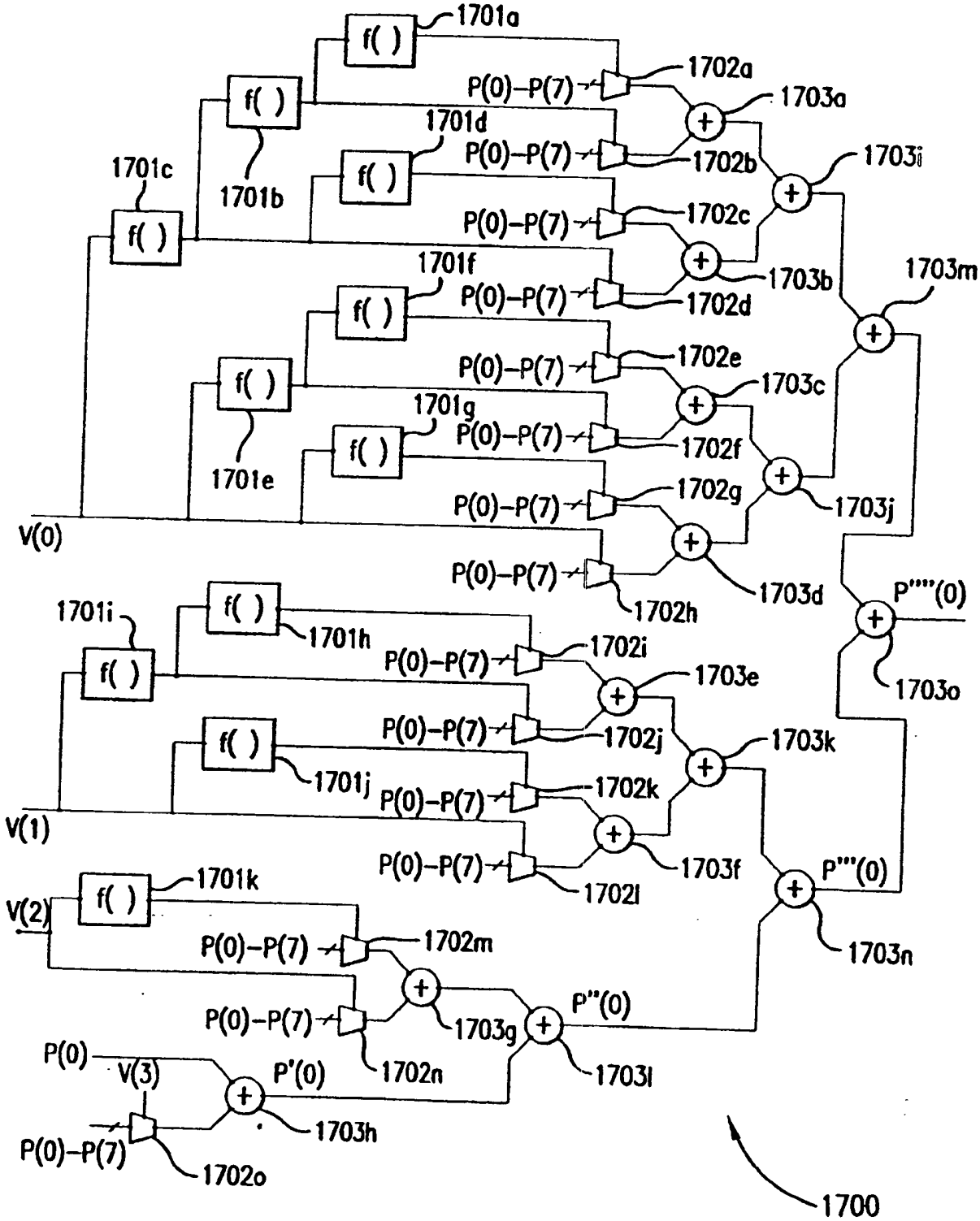

f( )
1701a
1701b
1701c
1701d
1701e
1701f
1701g
1701h
1701i
1701j
1701k
P(0)–P(7)
1702a
1702b
1702c
1702d
1702e
1702f
1702g
1702h
1702i
1702j
1702k
1702l
1702m
1702n
1702o
1703a
1703b
1703c
1703d
1703e
1703f
1703g
1703h
1703i
1703j
1703k
1703l
1703m
1703n
1703o
V(0)
V(1)
V(2)
V(3)
P(0)
P'(0)
P''(0)
P'''(0)
P''''(0)
1700

FIG. 28

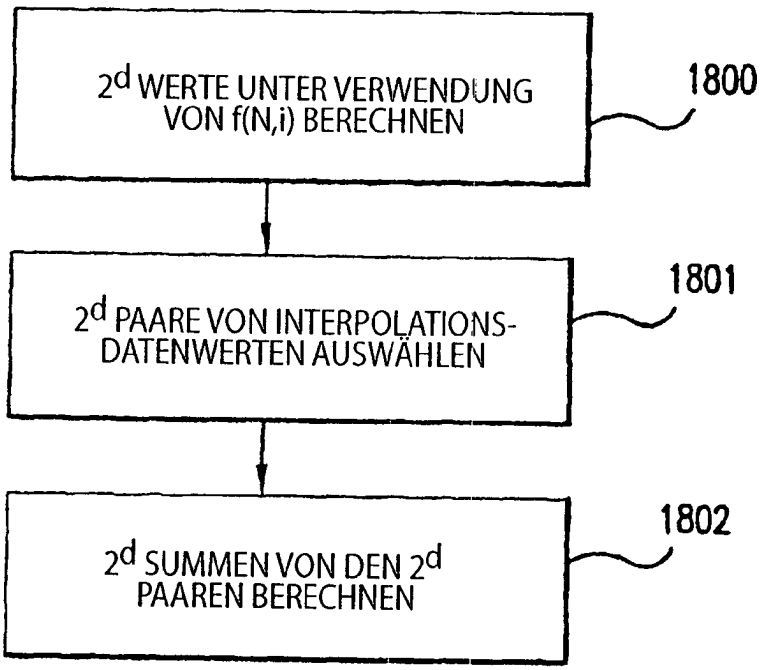

2^d WERTE UNTER VERWENDUNG VON f(N,i) BERECHNEN
1800
2^d PAARE VON INTERPOLATIONS-DATENWERTEN AUSWÄHLEN
1801
2^d SUMMEN VON DEN 2^d PAAREN BERECHNEN
1802

FIG.29

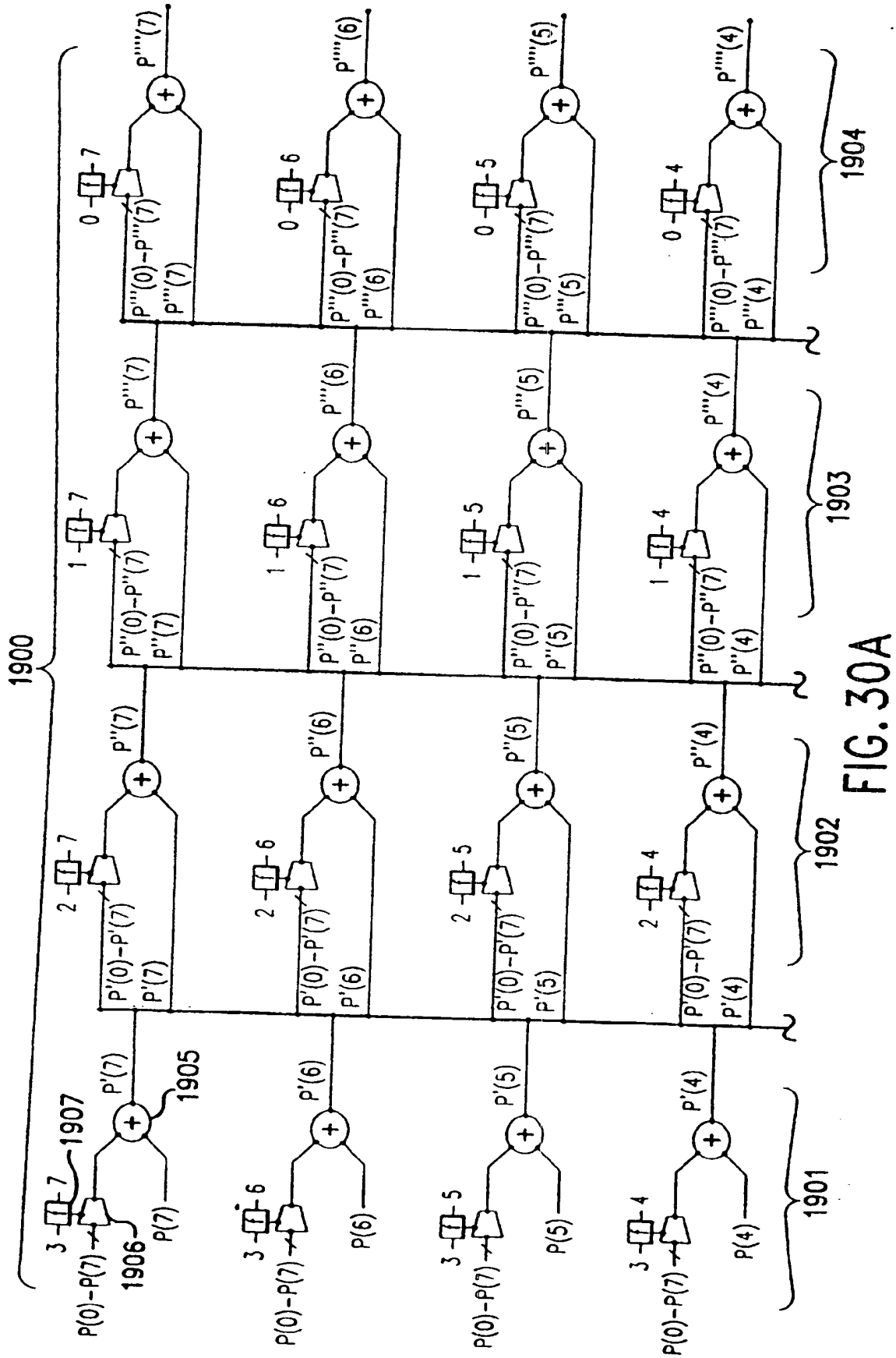

1900
1901
1902
1903
1904
1905
1906
1907
P(0)-P(7)
P(4)
P(5)
P(6)
P(7)
P'(4)
P'(5)
P'(6)
P'(7)
P''(4)
P''(5)
P''(6)
P''(7)
P'''(4)
P'''(5)
P'''(6)
P'''(7)
P''''(4)
P''''(5)
P''''(6)
P''''(7)

FIG. 30A

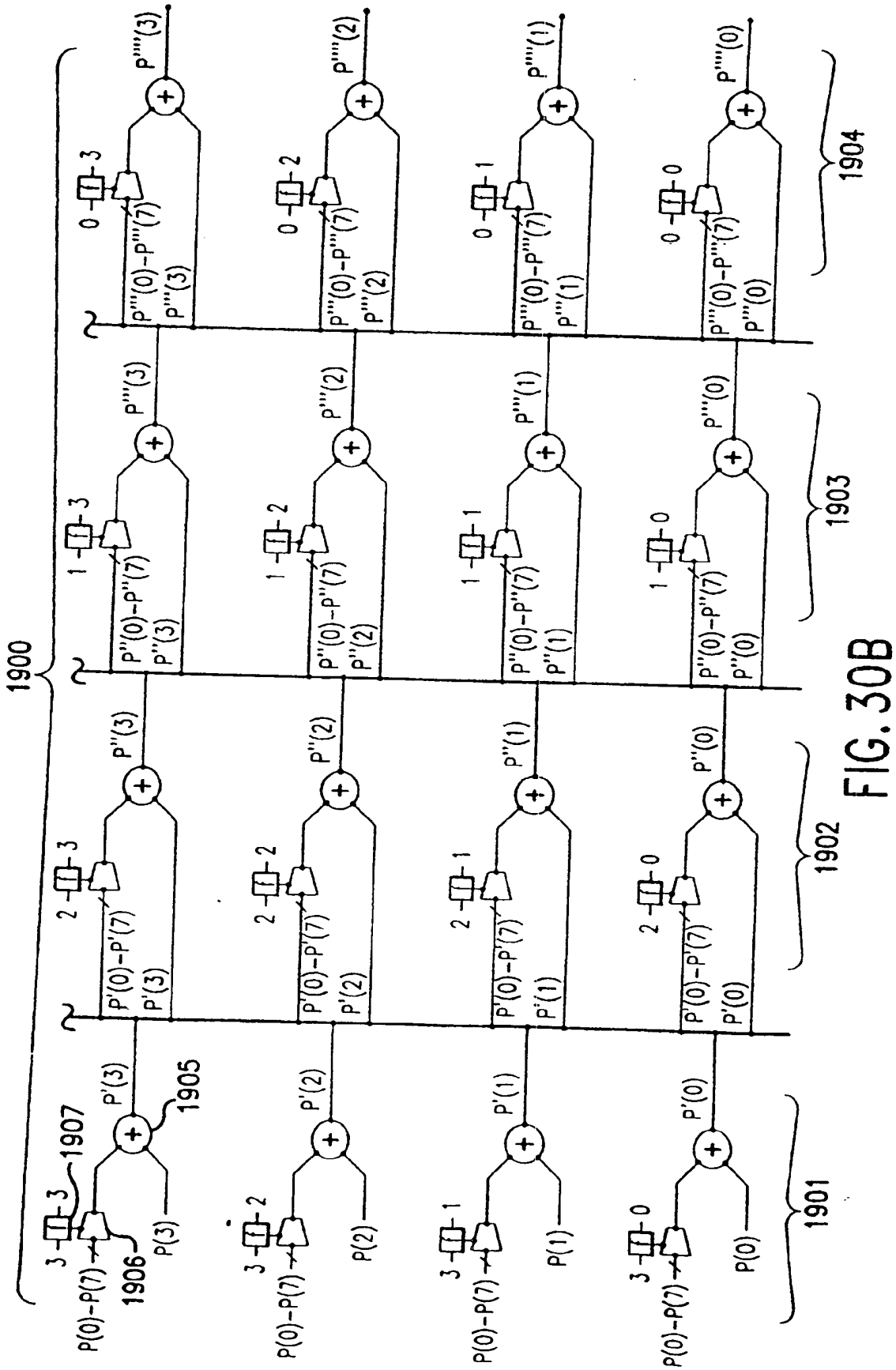

1900
1901
1902
1903
1904
1905
1906
1907
P(0)-P(7)
P(1)
P(2)
P(3)
P'(0)
P'(1)
P'(2)
P'(3)
P''(0)
P''(1)
P''(2)
P''(3)
P'''(0)
P'''(1)
P'''(2)
P'''(3)
P'(0)-P'(7)
P''(0)-P''(7)
P'''(0)-P'''(7)

FIG. 30B

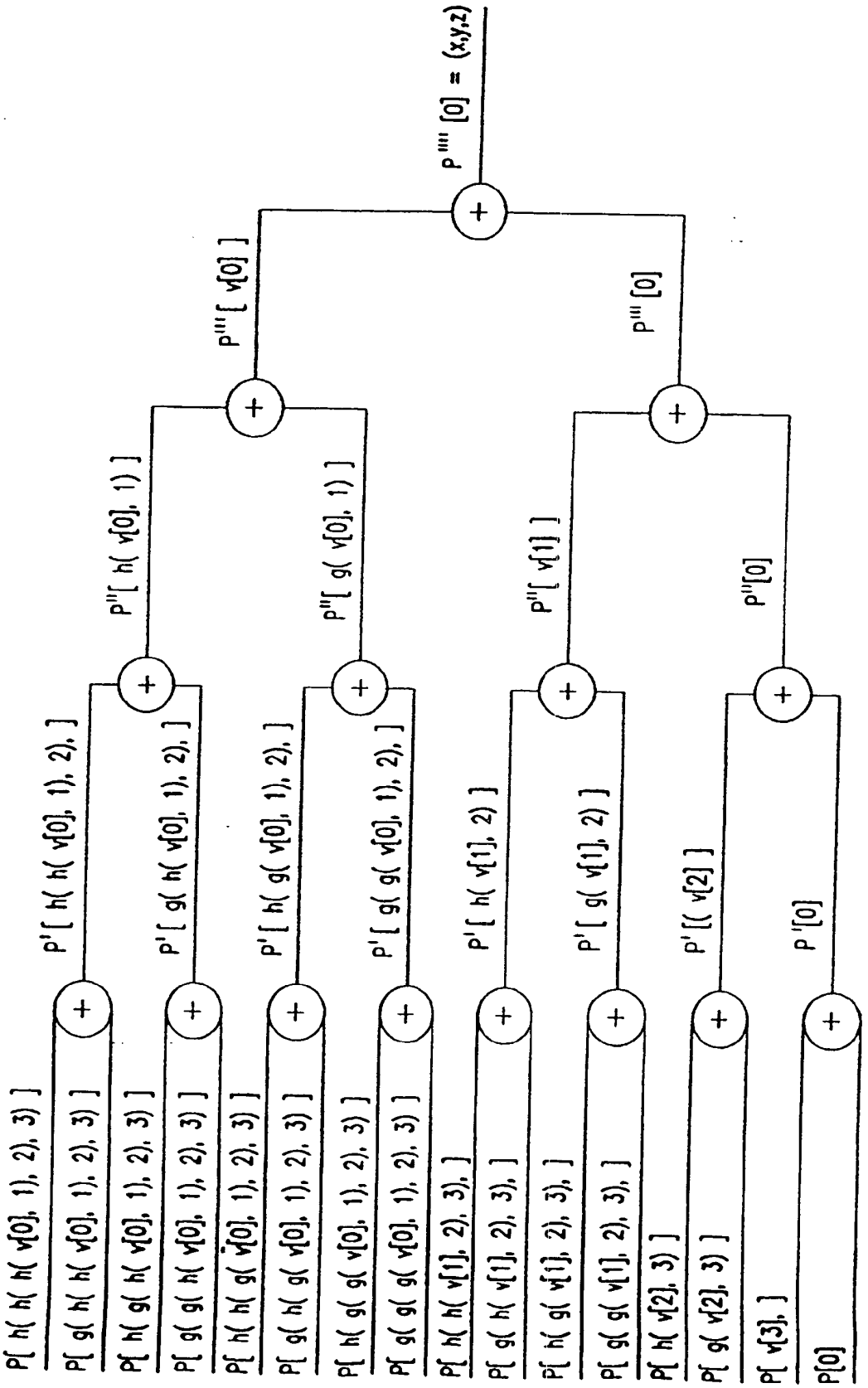

P'''' [0] = (x,y,z)
P''' [v[0]]
P''' [0]
P'' [h(v[0], 1)]
P'' [g(v[0], 1)]
P'' [v[1]]
P'' [0]
P' [h(h(v[0], 1), 2),]
P' [g(h(v[0], 1), 2),]
P' [h(g(v[0], 1), 2),]
P' [g(g(v[0], 1), 2),]
P' [h(v[1], 2),]
P' [g(v[1], 2),]
P' [(v[2]]
P' [0]
P [h(h(h(v[0], 1), 2), 3)]
P [g(h(h(v[0], 1), 2), 3)]
P [h(g(h(v[0], 1), 2), 3)]
P [g(g(h(v[0], 1), 2), 3)]
P [h(h(g(v[0], 1), 2), 3)]
P [g(h(g(v[0], 1), 2), 3)]
P [h(g(g(v[0], 1), 2), 3)]
P [g(g(g(v[0], 1), 2), 3)]
P [h(h(v[1], 2), 3),]
P [g(h(v[1], 2), 3),]
P [h(g(v[1], 2), 3),]
P [g(g(v[1], 2), 3),]
P [h(v[2], 3)]
P [g(v[2], 3)]
P [v[3],]
P [0]

FIG.31

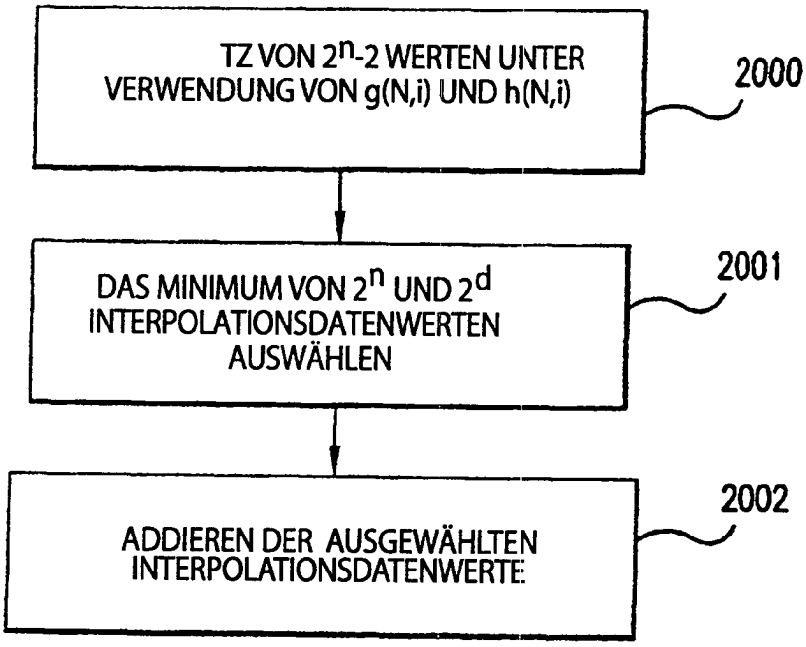

TZ VON 2^n-2 WERTEN UNTER VERWENDUNG VON g(N,i) UND h(N,i)
2000
DAS MINIMUM VON 2^n UND 2^d INTERPOLATIONSDATENWERTEN AUSWÄHLEN
2001
ADDIEREN DER AUSGEWÄHLTEN INTERPOLATIONSDATENWERTE
2002

FIG.32

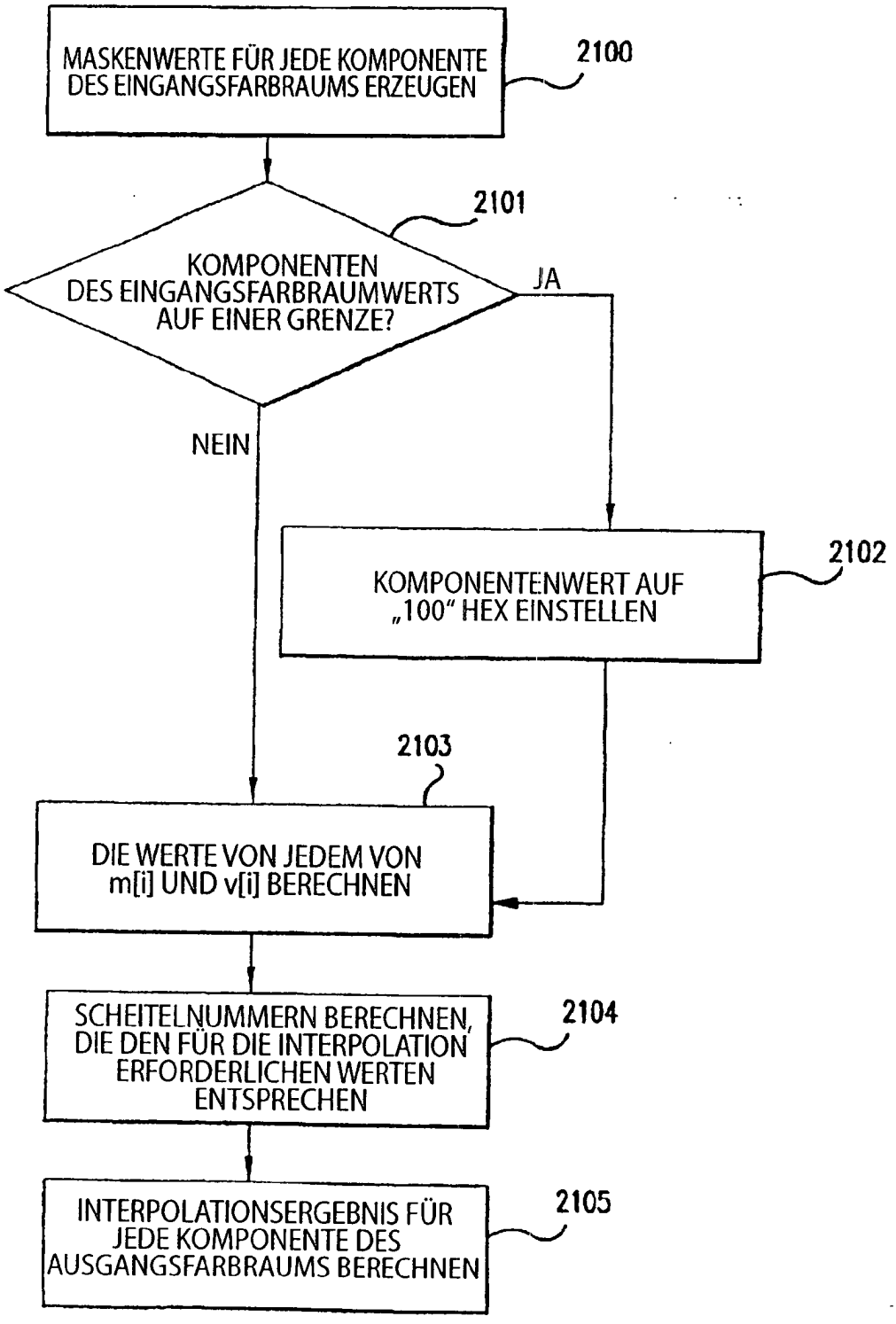

MASKENWERTE FÜR JEDE KOMPONENTE DES EINGANGSFARBRAUMS ERZEUGEN
2100
2101
KOMPONENTEN DES EINGANGSFARBRAUMWERTS AUF EINER GRENZE?
JA
NEIN
KOMPONENTENWERT AUF „100" HEX EINSTELLEN
2102
2103
DIE WERTE VON JEDEM VON m[i] UND v[i] BERECHNEN
SCHEITELNUMMERN BERECHNEN, DIE DEN FÜR DIE INTERPOLATION ERFORDERLICHEN WERTEN ENTSPRECHEN
2104
INTERPOLATIONSERGEBNIS FÜR JEDE KOMPONENTE DES AUSGANGSFARBRAUMS BERECHNEN
2105

FIG.33

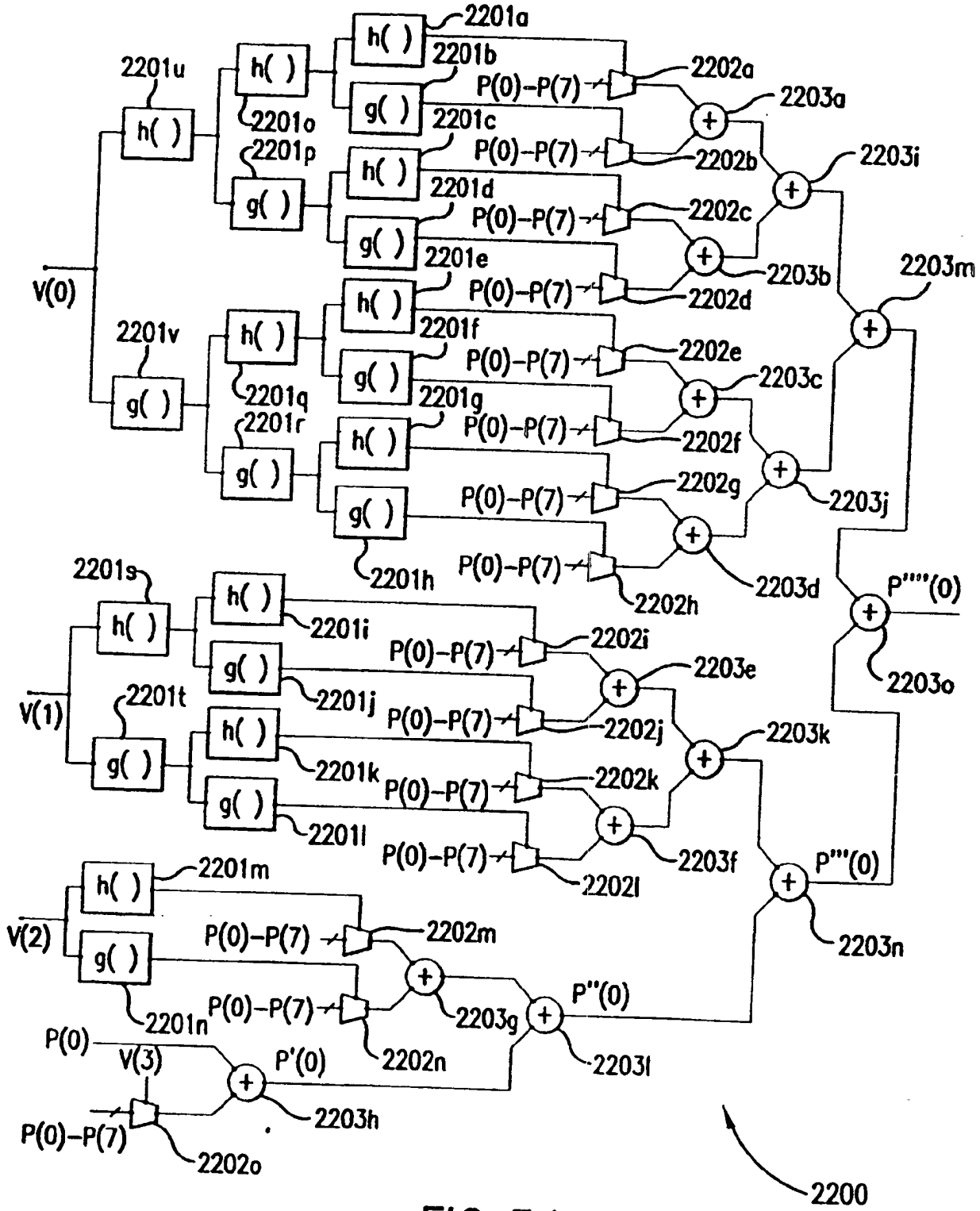

V(0)
V(1)
V(2)
V(3)
P(0)
h( )
g( )
2201a
2201b
2201c
2201d
2201e
2201f
2201g
2201h
2201i
2201j
2201k
2201l
2201m
2201n
2201o
2201p
2201q
2201r
2201s
2201t
2201u
2201v
P(0)–P(7)
2202a
2202b
2202c
2202d
2202e
2202f
2202g
2202h
2202i
2202j
2202k
2202l
2202m
2202n
2202o
2203a
2203b
2203c
2203d
2203e
2203f
2203g
2203h
2203i
2203j
2203k
2203l
2203m
2203n
2203o
P'(0)
P''(0)
P'''(0)
P''''(0)
2200

FIG. 34

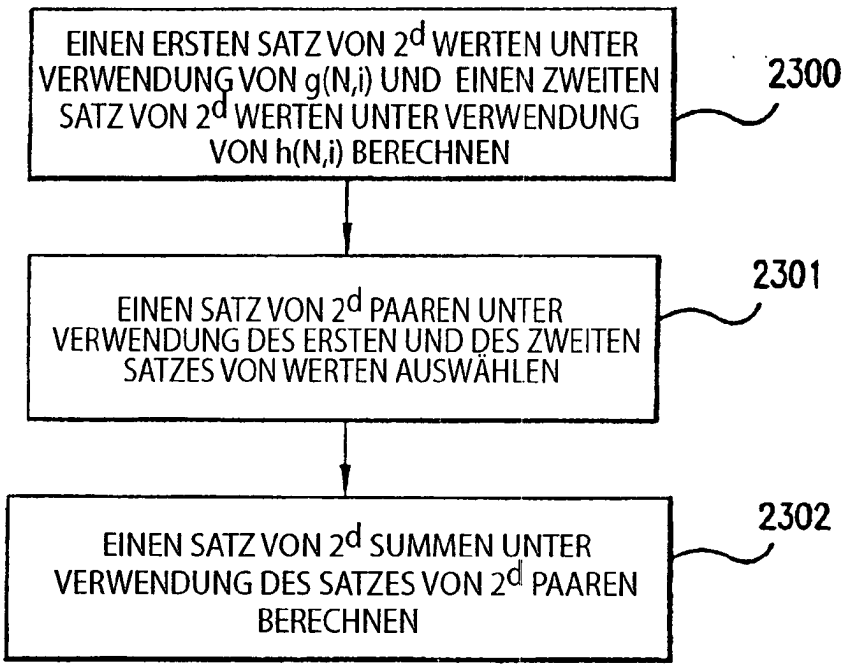

EINEN ERSTEN SATZ VON 2^d WERTEN UNTER VERWENDUNG VON g(N,i) UND EINEN ZWEITEN SATZ VON 2^d WERTEN UNTER VERWENDUNG VON h(N,i) BERECHNEN
2300
EINEN SATZ VON 2^d PAAREN UNTER VERWENDUNG DES ERSTEN UND DES ZWEITEN SATZES VON WERTEN AUSWÄHLEN
2301
EINEN SATZ VON 2^d SUMMEN UNTER VERWENDUNG DES SATZES VON 2^d PAAREN BERECHNEN
2302

FIG.35

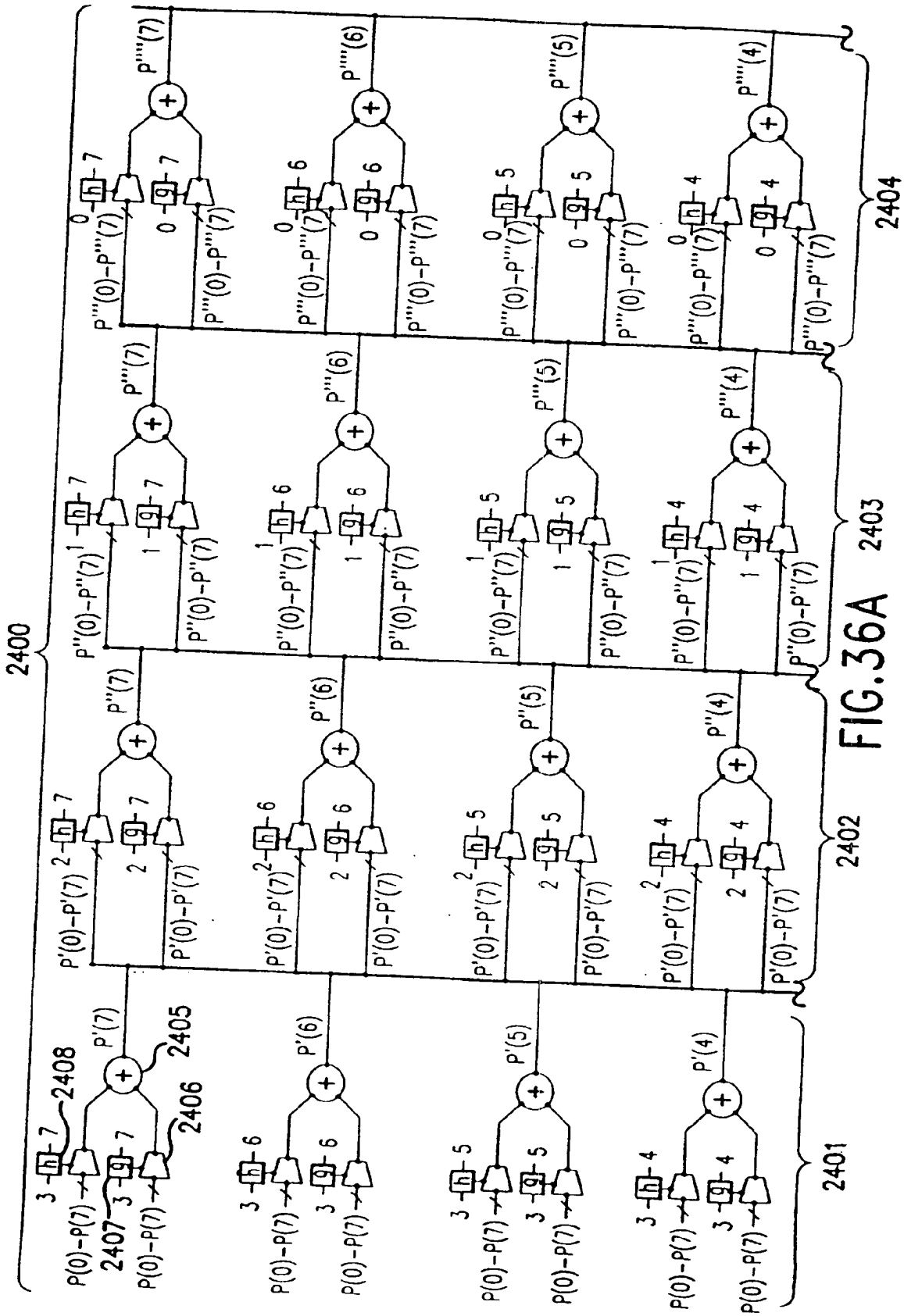

2400
2401
2402
2403
2404
2405
2406
2407
2408
P(0)-P(7)
P'(0)-P'(7)
P''(0)-P''(7)
P'''(0)-P'''(7)
P'(7)
P'(6)
P'(5)
P'(4)
P''(7)
P''(6)
P''(5)
P''(4)
P'''(7)
P'''(6)
P'''(5)
P'''(4)

FIG.36A

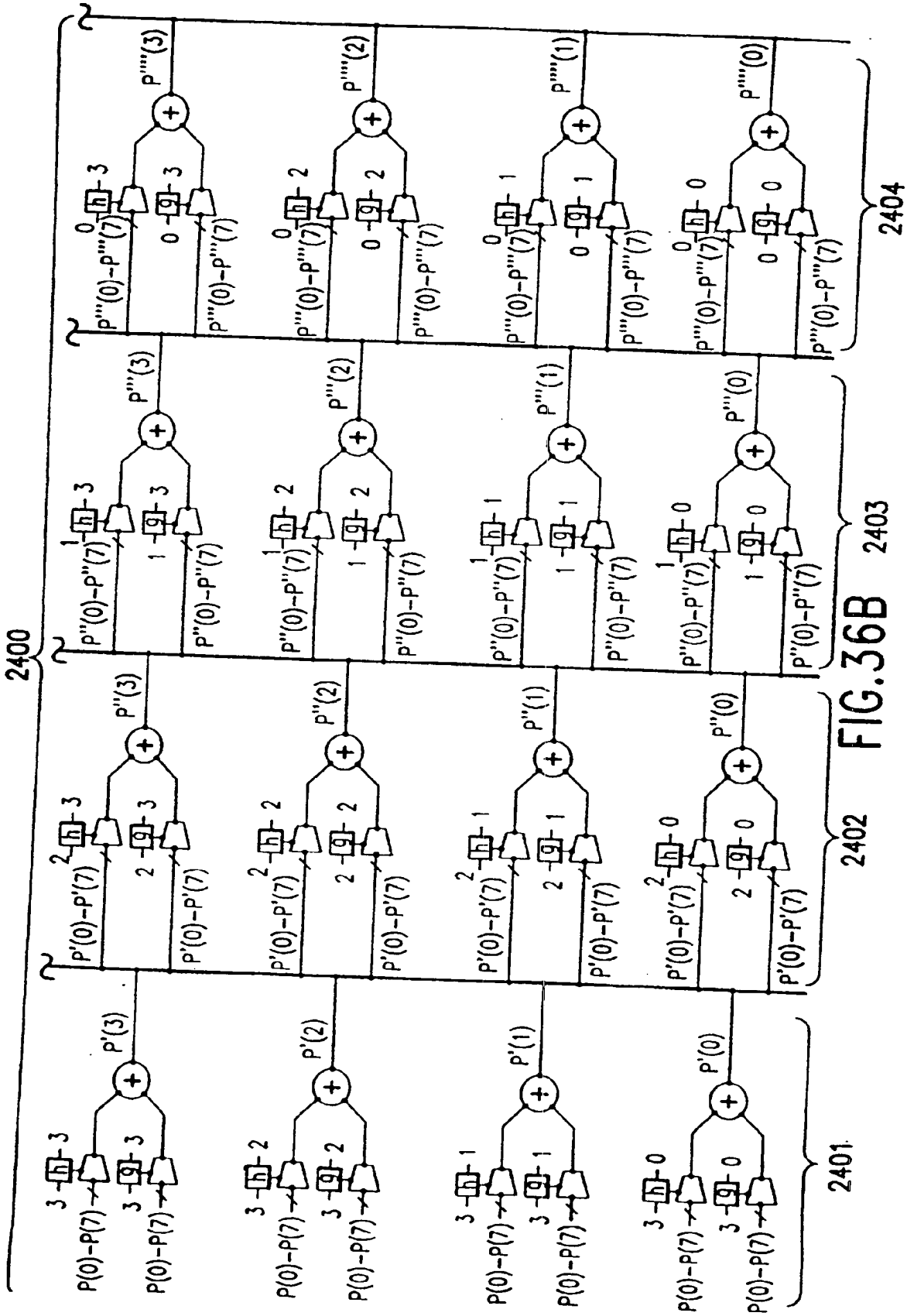

2400
2401
2402
2403
2404
P(0)-P(7)
P'(0)-P'(7)
P''(0)-P''(7)
P'''(0)-P'''(7)
P'(3)
P'(2)
P'(1)
P'(0)
P''(3)
P''(2)
P''(1)
P''(0)
P'''(3)
P'''(2)
P'''(1)
P'''(0)

FIG.36B

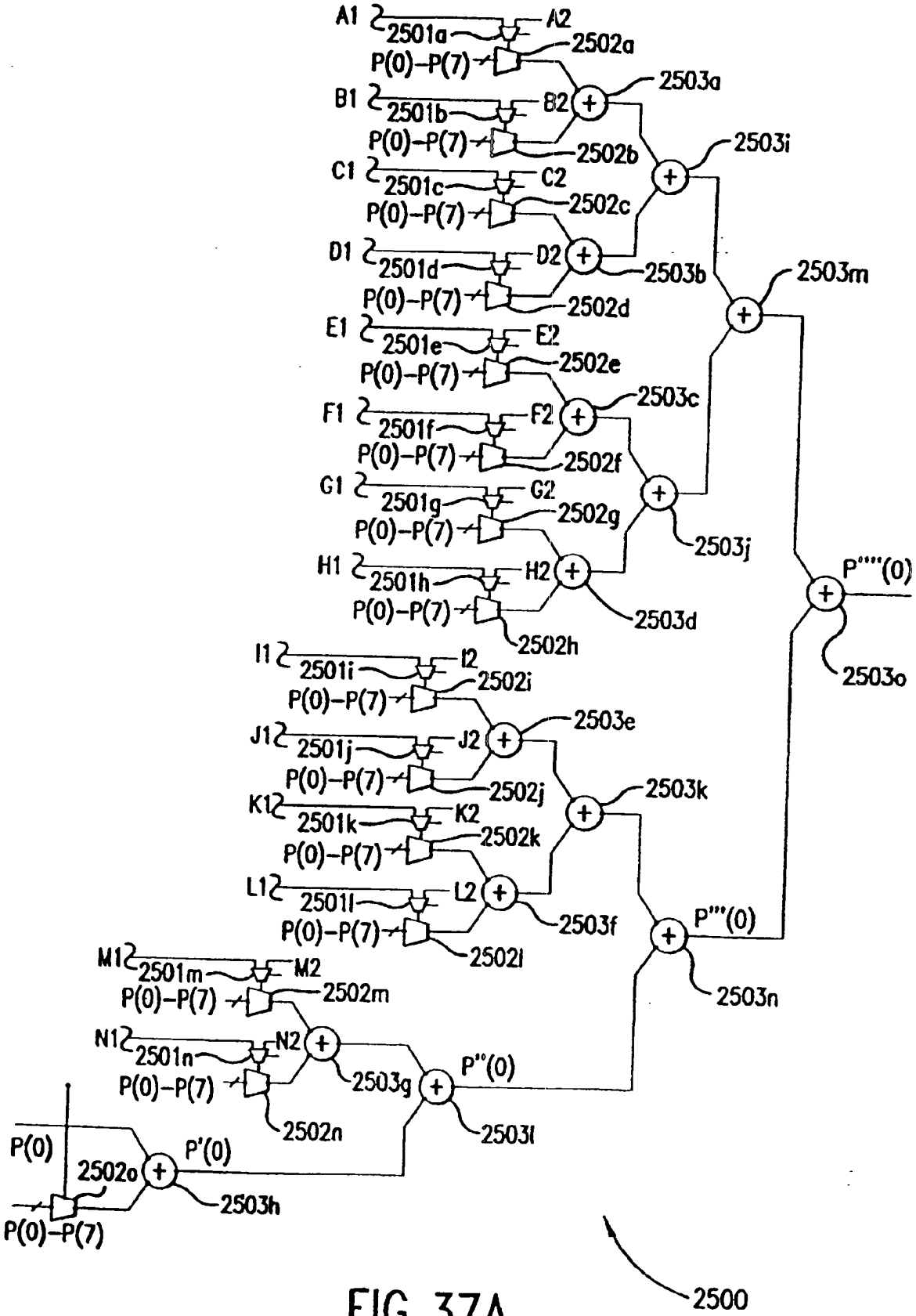

A1
2501a
A2
2502a
P(0)–P(7)
2503a
B1
2501b
B2
P(0)–P(7)
2502b
2503i
C1
2501c
C2
2502c
P(0)–P(7)
D1
2501d
D2
2503b
2503m
P(0)–P(7)
2502d
E1
2501e
E2
2502e
P(0)–P(7)
2503c
F1
2501f
F2
P(0)–P(7)
2502f
G1
2501g
G2
2502g
P(0)–P(7)
2503j
P''''(0)
H1
2501h
H2
P(0)–P(7)
2503d
2502h
2503o
I1
2501i
I2
2502i
P(0)–P(7)
2503e
J1
2501j
J2
P(0)–P(7)
2502j
2503k
K1
2501k
K2
2502k
P(0)–P(7)
L1
2501l
L2
2503f
P'''(0)
P(0)–P(7)
2502l
M1
2501m
M2
2502m
P(0)–P(7)
2503n
N1
2501n
N2
2503g
P''(0)
P(0)–P(7)
2502n
2503l
P(0)
2502o
P'(0)
2503h
P(0)–P(7)
2500

FIG. 37A

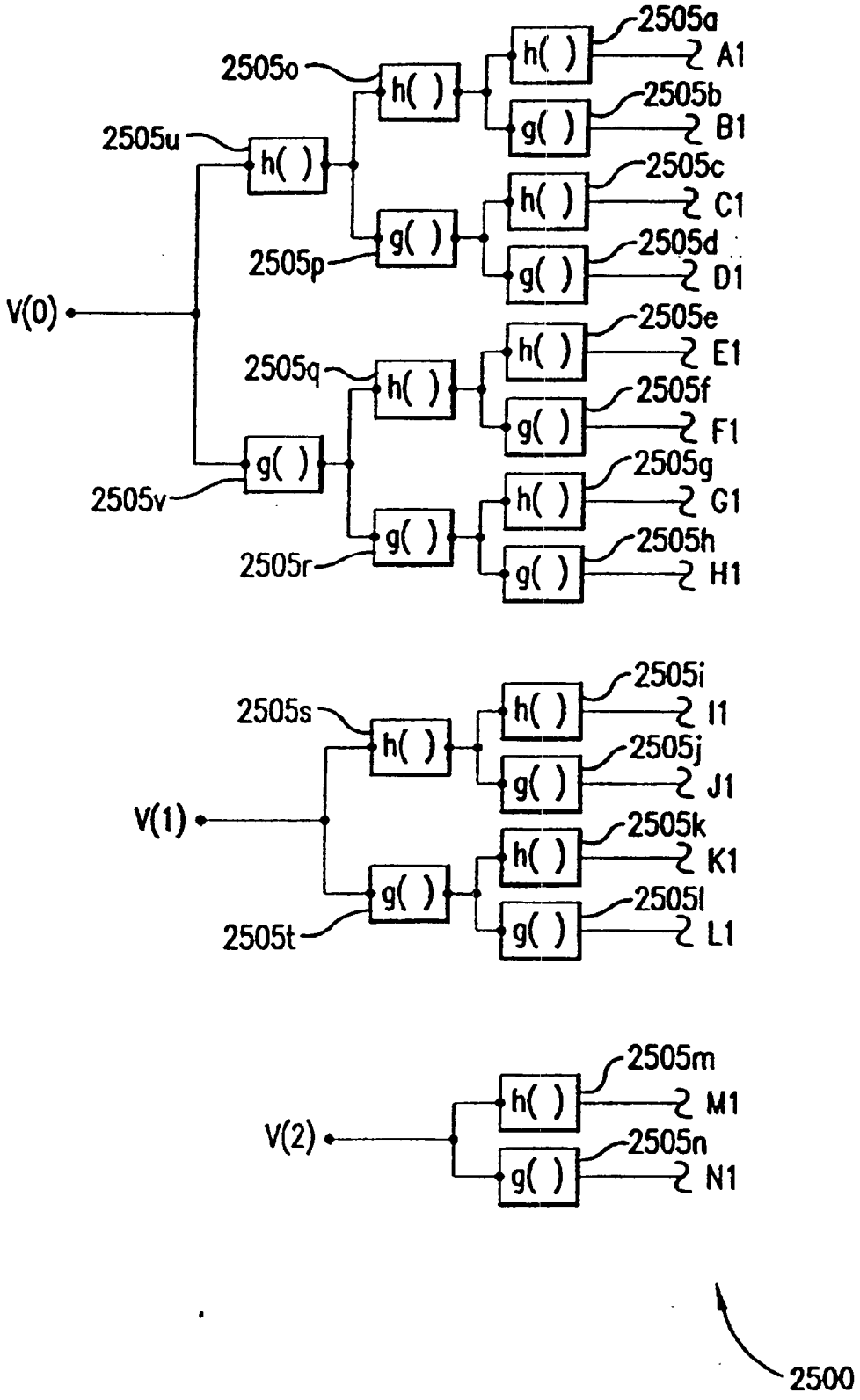

V(0)
V(1)
V(2)
2505a
A1
2505b
B1
2505c
C1
2505d
D1
2505e
E1
2505f
F1
2505g
G1
2505h
H1
2505i
I1
2505j
J1
2505k
K1
2505l
L1
2505m
M1
2505n
N1
2505o
2505p
2505q
2505r
2505s
2505t
2505u
2505v
h( )
g( )
2500

FIG. 37B

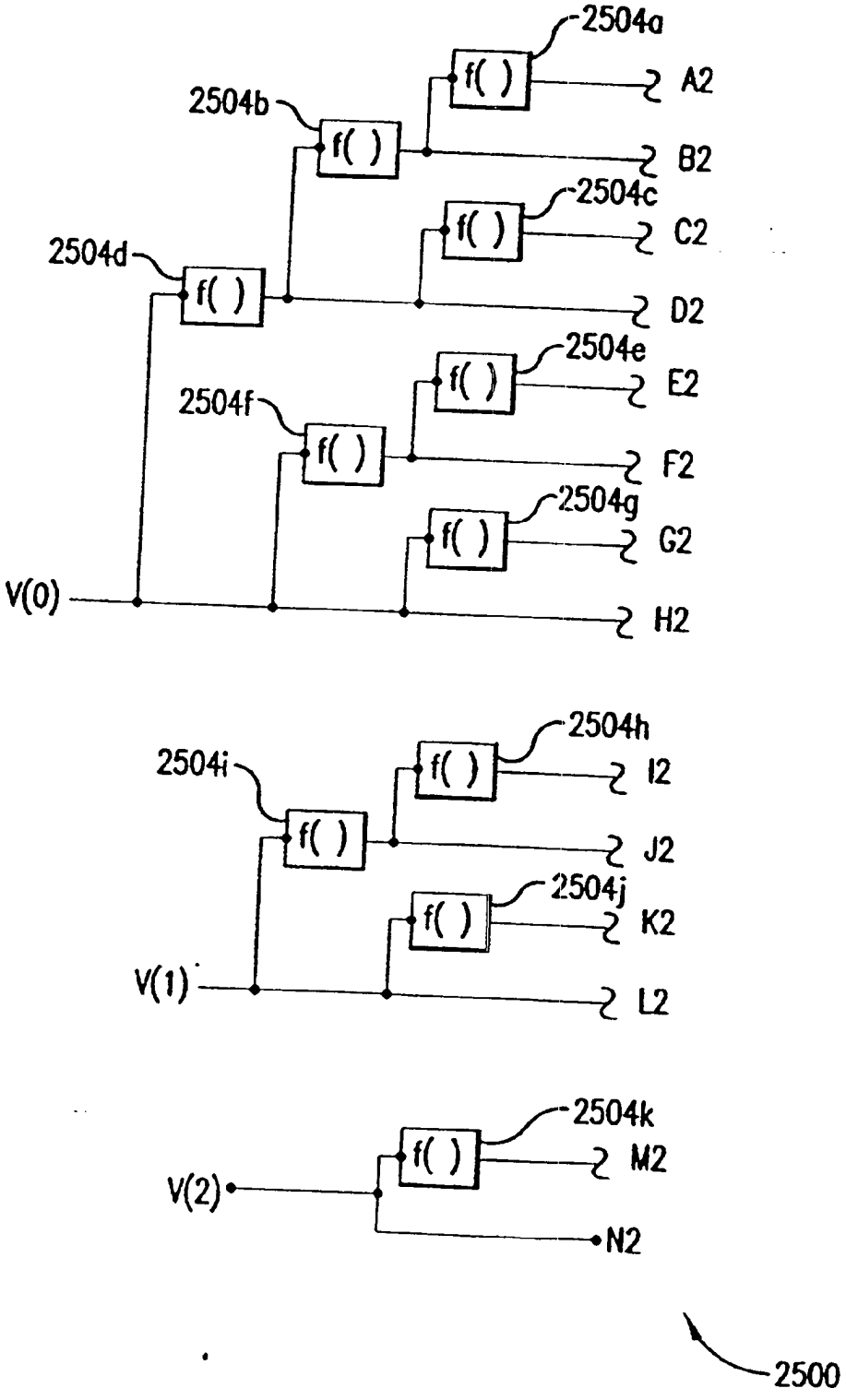

2504a
f( )
A2
2504b
f( )
B2
2504c
f( )
C2
2504d
f( )
D2
2504e
f( )
E2
2504f
f( )
F2
2504g
f( )
G2
V(0)
H2
2504h
f( )
I2
2504i
f( )
J2
2504j
f( )
K2
V(1)
L2
2504k
f( )
M2
V(2)
N2
2500

FIG. 37C

```
#define C_INC     1
#define B_INC     C_INC*17
#define A_INC     B_INC*17
#define TABLE_INC A_INC*17
/*****************************************/
/** radial_interpolation()              **/
/*****************************************/
void radial_interpolation(unsigned char input[],
unsigned char table[][TABLE_INC], unsigned char output[])
{
    register int        origin, v0, v1, v2, v3;
    register int        a, b, c ;
    register int        x, y, z ;
    /*************************************/
    /* Snap max value to last table value */
    /*************************************/
    a = ( input[0] == 0x0ff ) ? 0x100 : (int) input[0] ;
    b = ( input[1] == 0x0ff ) ? 0x100 : (int) input[1] ;
    c = ( input[2] == 0x0ff ) ? 0x100 : (int) input[2] ;
    /*************************************/
    /* Compute indexes into table        */
    /*************************************/
    origin = A_INC*(a >> 4)
           + B_INC*(b >> 4)
           +       (c >> 4) ;
    v0 = origin + A_INC*( (a & 0x01) )
                + B_INC*( (b & 0x01) )
                +         (c & 0x01) ) ;
    v1 = origin + A_INC*( (a & 0x02) >> 1 )
                + B_INC*( (b & 0x02) >> 1 )
                +         (c & 0x02 ) >> 1) ;
    v2 = origin + A_INC*( (a & 0x04) >> 2 )
                + B_INC*( (b & 0x04) >> 2 )
                +         (c & 0x04) >> 2) ;
    v3 = origin + A_INC*( (a & 0x08) >> 3 )
                + B_INC*( (b & 0x08) >> 3 )
                +         (c & 0x08) >> 3 ) ;
    /*************************************/
    /* Compute x */
    /*************************************/
    x = ( ( (int) table[0][origin]
        +   (int) table[0][v0]
        + ( (int) table[0][v1] << 1 )
        + ( (int) table[0][v2] << 2 )
        + ( (int) table[0][v3] << 3 )
        +   0x08 ) >> 4 ) ;          /* Round */
    /*************************************/
    /* Compute y */
    /*************************************/
    y = ( ( (int) table[1][origin]
        +   (int) table[1][v0]
        + ( (int) table[1][v1] << 1 )
        + ( (int) table[1][v2] << 2 )
        + ( (int) table[1][v3] << 3 )
        +   0x08 ) >> 4 ) ;          /* Round */
    /*************************************/
    /* Compute z */
    /*************************************/
    z = ( ( (int) table[2][origin]
        +   (int) table[2][v0]
        + ( (int) table[2][v1] << 1 )
        + ( (int) table[2][v2] << 2 )
        + ( (int) table[2][v3] << 3 )
        +   0x08 ) >> 4 ) ;          /* Round */
    output[0] = (char) x ;
    output[1] = (char) y ;
    output[2] = (char) z ;
}
```

FIG.38

```
-----------------------------------------------
 Radial Interpolation.                        --
-----------------------------------------------
entity radial is
port (
     -- Input Pixel (a,b,c) -- :
     input:          in    INPUT_ARRAY ;
     -- Vertex Points of Cube from Color Table --
     point:          in    POINT_ARRAY ;
     -- Output Pixel (x,y,z) --
     output:         out   OUTPUT_ARRAY
) ;
end radial ;
-----------------------------------------
architecture radialrtl of radial is
-----------------------------------------
     signal  vertex:       VERTEX_ARRAY ;
     signal  point_value:  POINT_VALUE_ARRAY ;
begin
-----------------------------------------------------
-- Compute selected vertices from slices across input.
-----------------------------------------------------
VERTICIES: process ( input )
begin
     ---------------------------------------------
     -- For each slice across the input lower bits --
     ---------------------------------------------
     for i in NUM_INTERP_BITS-1 downto 0 loop
          vertex(i) <= input(0)(i) & input(1)(i)
                                   & input(2)(i);
     end loop ;
end process ;
-----------------------------------------------
-- Select the values of vertices to be used by --
-- interpolation.  The last two vertices are   --
-- always point(0) and the rounding constant.  --
-----------------------------------------------
MUX: process ( vertex, point )
begin
```

```
-----------------------------
-- For each output dimension --
-----------------------------
for d in 0 to NUM_OUTPUT_DIMEN-1 loop
   ---------------
   -- Initialize --
   ---------------
   point_value(d) <= (others => (others => '0'));
   ---------------------------------------
   -- point_value( 0 : NUM_INTERP_BITS-1 ) --
   ---------------------------------------
   for i in 0 to NUM_INTERP_BITS-1 loop
     case vertex(i) is
        when "000" => point_value(d)(i)(7+i downto i)
                                  <= point(d)(0) ;
        when "001" => point_value(d)(i)(7+i downto i
                                  <= point(d)(1) ;
        when "010" => point_value(d)(i)(7+i downto i)
                                  <= point(d)(2) ;
        when "011" => point_value(d)(i)(7+i downto i)
                                  <= point(d)(3) ;
        when "100" => point_value(d)(i)(7+i downto i)
                                  <= point(d)(4) ;
        when "101" => point_value(d)(i)(7+i downto i)
                                  <= point(d)(5);
        when "110" => point_value(d)(i)(7+i downto i)
                                  <= point(d)(6);
        when others => point_value(d)(i)(7+i downto!i)
                                  <= point(d)(7);
     end case ;
```

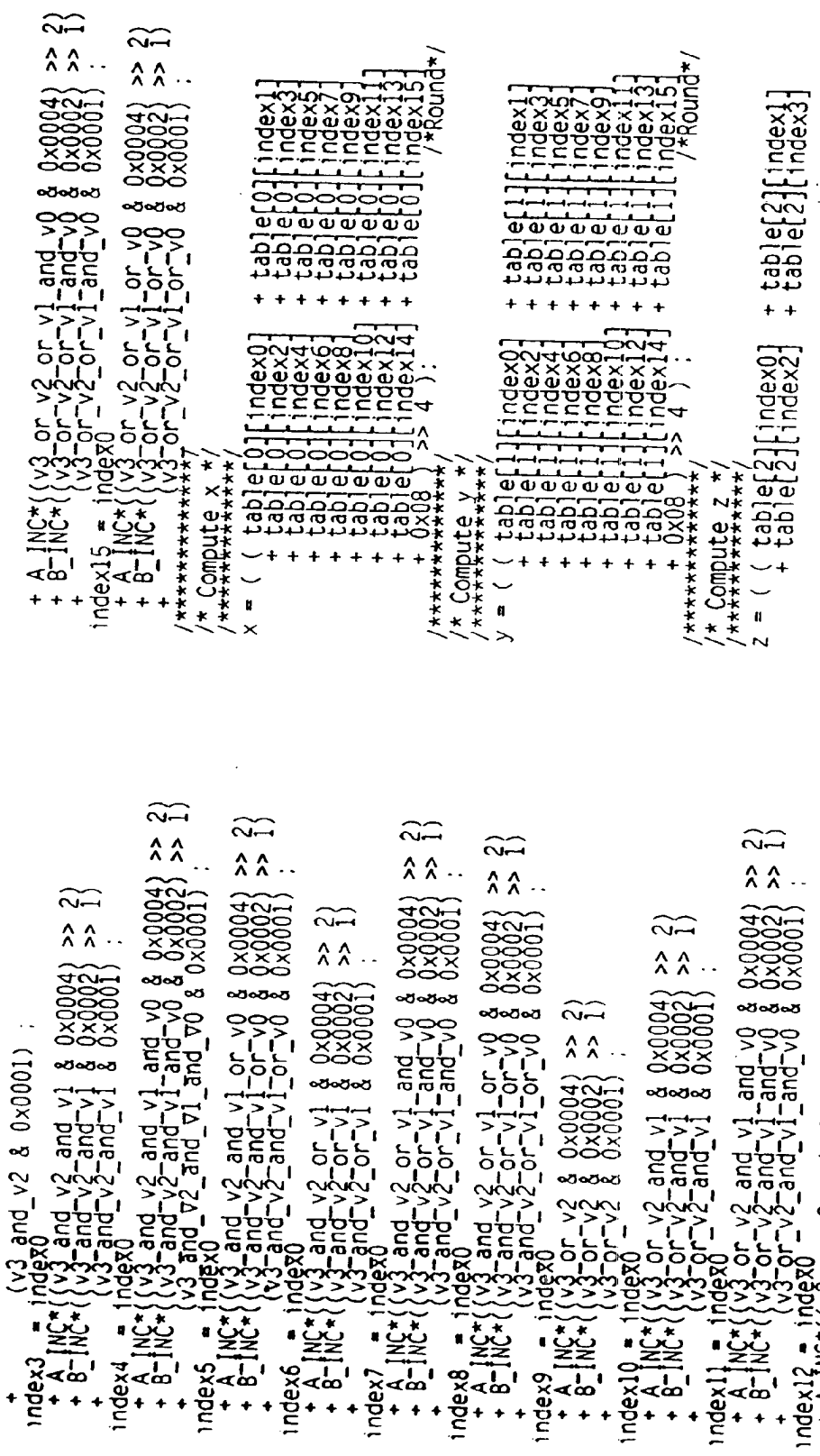
FIG.39A

```
        ---------------
        -- point(d)(0) --
        ---------------
        point_value(d)(NUM_INTERP_BITS)(7 downto 0)
                                        <= point(d)(0) ;
        ---------------------
        -- Rounding constant --
        ---------------------
        point_value(d)(NUM_INTERP_BITS+1)
                        (NUM_INTERP_BITS-1) <= '1' ;
      end loop ;
    end loop ;
end process ;
-------------------------------------------------
-- SUM adds the selected vertices values and trun- --
-- cates to eight bits. The process is implemented --
-- to minimize time by successive pairing and      --
-- adding.                                         --
-------------------------------------------------
SUM: process ( point_value )
    variable  temp:       POINT_VALUES ;
    variable  bound:      INTEGER ;
begin
    ----------------------------
    -- For each output dimension --
    ----------------------------
    for d in 0 to NUM_OUTPUT_DIMEN-1 loop
      ----------------------------
      -- Initialize temp and bound. --
      ----------------------------
      temp := point_value(d) ;
      bound := NUM_INTERP_BITS + 2 ;
      ---------------------------------------
      -- Loop for ceil(log2(NUM_INTERP_BITS-1)). --
      ---------------------------------------
      for i in 0 to NUM_INTERP_BITS-1 loop
        ---------------------------------
        -- Pair temp array elements and add. --
        ---------------------------------
        for j in 0 to (NUM_INTERP_BITS/2) loop
          ----------------------------
          -- Step 1: Move temp(even's) --
          ----------------------------
          if ( 2*j < bound ) then
              temp(j) := temp(2*j) ;
          else
              temp(j) := (others => '0') ;
          end if ;
          ----------------------------
          -- Step 2: Add temp(odd's) --
          ----------------------------
          if ( (2*j)+1 < bound ) then
              temp(j) := temp(j)
                         + temp((2*j)+1);
          else
              temp(j) := temp(j) ;
          end if ;
        end loop ;
        -------------------------------
        -- Set bound for next iteration --
        -------------------------------
        bound := ( bound / 2 ) + ( bound mod 2 ) ;
      end loop ;
      ---------------------
      -- Truncate output. --
      ---------------------
      output(d) <= temp(0)(7+NUM_INTERP_BITS
                           downto NUM_INTERP_BITS) ;
    end loop ;
end process ;
-------------------------------------------------
end radialrtl;
-------------------------------------------------
```

FIG.39B

```
#define C_INC      1
#define B_INC      C_INC*17
#define A_INC      B_INC*17
#degine TABLE_INC A_INC*17

/*************************************************/
/** pruned tetrahedral interpolation()          **/
/*************************************************/

void pruned_tetrahedral_interpolation(
  unsigned char input[], unsigned char table[][TABLE_INC],
  unsigned char output[] )
{
register int  a, b, c, x, y, z ;
register char v0, v1, v2, v3;
register char v1_and_v0, v2_and_v1, v3_and_v2 ;
register char v1_or_v0,  v2_or_v1,  v3_or_v2 ;
register char v2_and_v1_and_v0, v3_or_v2_or_v1_or_v0;
register char v2_and_v1_or_v0, v3_or_v2_or_v1_and_v0;
register char v2_or_v1_and_v0, v3_or_v2_and_v1_or_v0;
register char v2_or_v1_or_v0, v3_or_v2 and_v1_and_v0;
register char v3_and_v2_and_v0, v3_and_v2_or_v1_or_v0;
register char v3_and_v2_or_v0, v3_and_v2_or_v1_and_v0;
register char v3_or_v2_and_v0, v3_and_v2_and_v1 or_v0;
register char v3_or_v2_or_v0, v3_and_v2_and_v1_and_v0;
register int  index0,  index1,  index2,  index3 ;
register int  index4,  index5,  index6,  index7 ;
register int  index8,  index9,  index10, index11 ;
register int  index12, index13, index14, index15 ;

  /*************************************/
  /* Snap last value to last table value */
  /*************************************/

  a = ( input[0] == 0x0ff ) ? 0x100 : (int) input[0] ;
  b = ( input[1] == 0x0ff ) ? 0x100 : (int) input[1] ;
  c = ( input[2] == 0x0ff ) ? 0x100 : (int) input[2] ;

  /*********************************************/
  /* Compute slices across the input components */
  /*********************************************/

  v0 = ( (a & 0x01) << 2 )
     + ( (b & 0x01) << 1 ) +   (c & 0x01) ;
  v1 = ( (a & 0x02) << 1 )
     +   (b & 0x02)        + ( (c & 0x02) >> 1 ) ;
  v2 =   (a & 0x04)
     + ( (b & 0x04) >> 1 ) + ( (c & 0x04) >> 2 ) ;
  v3 = ( (a & 0x08) >> 1 )
     + ( (b & 0x08) >> 2 ) + ( (c & 0x08) >> 3 ) ;

  /*********************************/
  /* Compute offset from origin. */
  /*********************************/

  v1_and_v0                = v1 & v0 ;
  v1_or_v0                 = v1 | v0 ;
  v2_and_v1                = v2 & v1 ;
  v2_and_v1_and_v0         = v2 & v1_and_v0 ;
  v2_and_v1_or_v0          = v2 & v1_or_v0 ;
  v2_or_v1                 = v2 | v1 ;
  v2_or_v1_and v0          = v2 | v1_and_v0 ;
  v2_or_v1_or_v0           = v2 | v1_or_v0 ;
  v3_and_v2                = v3 & v2 ;
  v3_and_v2_and_v1         = v3 & v2_and_v1 ;
  v3_and_v2_and_v1_and_v0  = v3 & v2_and_v1_and_v0 ;
  v3_and_v2_and_v1_or_v0   = v3 & v2_and_v1_or_v0 ;
  v3_and_v2_or_v1          = v3 & v2_or_v1 ;
  v3_and_v2_or_v1_and_v0   = v3 & v2_or_v1_and_v0 ;
  v3_and_v2_or_v1_or_v0    = v3 & v2_or_v1_or_v0 ;
  v3_or_v2                 = v3 | v2 ;
  v3_or_v2_and_v1          = v3 | v2_and_v1 ;
  v3_or_v2_and_v1_and_v0   = v3 | v2_and_v1_and_v0 ;
  v3_or_v2_and_v1_or_v0    = v3 | v2_and_v1_or_v0 ;
  v3_or_v2_or_v1           = v3 | v2_or_v1 ;
  v3_or_v2_or_v1_and_v0    = v3 | v2_or_v1_and_v0 ;
  v3_or_v2_or_v1_or_v0     = v3 | v2_or_v1_or_v0 ;

  /*********************************/
  /* Generate indices into table */
  /*********************************/

  index0  = ( (a >> 4) * A_INC )
    + ( (b >> 4) * B_INC ) + (c >> 4) ;
  index1  = index0
    + A_INC*((v3 & 0x0004) >> 2)
    + B_INC*((v3 & 0x0002) >> 1)
    +        (v3 & 0x0001) ;
  index2  = index0
    + A_INC*((v3_and_v2 & 0x0004) >> 2)
    + B_INC*((v3_and_v2 & 0x0002) >> 1)
```

FIG.40A



```
  +          (v3_and_v2 & 0x0001) ;
index3  = index0
  + A_INC*((v3_and_v2_and_v1 & 0x0004) >> 2)
  + B_INC*((v3_and_v2_and_v1 & 0x0002) >> 1)
  +          (v3_and_v2_and_v1 & 0x0001) ;
index4  = index0
  + A_INC*((v3_and_v2_and_v1_and_v0 & 0x0004) >> 2)
  + B_INC*((v3_and_v2_and_v1_and_v0 & 0x0002) >> 1)
  +          (v3_and_v2_and_v1_and_v0 & 0x0001) ;
index5  = index0
  + A_INC*((v3_and_v2_and_v1_or_v0 & 0x0004) >> 2)
  + B_INC*((v3_and_v2_and_v1_or_v0 & 0x0002) >> 1)
  +          (v3_and_v2_and_v1_or_v0 & 0x0001) ;
index6  = index0
  + A_INC*((v3_and_v2_or_v1 & 0x0004) >> 2)
  + B_INC*((v3_and_v2_or_v1 & 0x0002) >> 1)
  +          (v3_and_v2_or_v1 & 0x0001) ;
index7  = index0
  + A_INC*((v3_and_v2_or_v1_and_v0 & 0x0004) >> 2)
  + B_INC*((v3_and_v2_or_v1_and_v0 & 0x0002) >> 1)
  +          (v3_and_v2_or_v1_and_v0 & 0x0001) ;
index8  = index0
  + A_INC*((v3_and_v2_or_v1_or_v0 & 0x0004) >> 2)
  + B_INC*((v3_and_v2_or_v1_or_v0 & 0x0002) >> 1)
  +          (v3_and_v2_or_v1_or_v0 & 0x0001) ;
index9  = index0
  + A_INC*((v3_or_v2 & 0x0004) >> 2)
  + B_INC*((v3_or_v2 & 0x0002) >> 1)
  +          (v3_or_v2 & 0x0001) ;
index10 = index0
  + A_INC*((v3_or_v2_and_v1 & 0x0004) >> 2)
  + B_INC*((v3_or_v2_and_v1 & 0x0002) >> 1)
  +          (v3_or_v2_and_v1 & 0x0001) ;
index11 = index0
  + A_INC*((v3_or_v2_and_v1_and_v0 & 0x0004) >> 2)
  + B_INC*((v3_or_v2_and_v1_and_v0 & 0x0002) >> 1)
  +          (v3_or_v2_and_v1_and_v0 & 0x0001) ;
index12 = index0
  + A_INC*((v3_or_v2_and_v1_or_v0 & 0x0004) >> 2)
  + B_INC*((v3_or_v2_and_v1_or_v0 & 0x0002) >> 1)
  +          (v3_or_v2_and_v1_or_v0 & 0x0001) ;
index13 = index0
  + A_INC*((v3_or_v2_or_v1 & 0x0004) >> 2)
  + B_INC*((v3_or_v2_or_v1 & 0x0002) >> 1)
  +          (v3_or_v2_or_v1 & 0x0001) ;
index14 = index0
  + A_INC*((v3_or_v2_or_v1_and_v0 & 0x0004) >> 2)
  + B_INC*((v3_or_v2_or_v1_and_v0 & 0x0002) >> 1)
  +          (v3_or_v2_or_v1_and_v0 & 0x0001) ;
index15 = index0
  + A_INC*((v3_or_v2_or_v1_or_v0 & 0x0004) >> 2)
  + B_INC*((v3_or_v2_or_v1_or_v0 & 0x0002) >> 1)
  +          (v3_or_v2_or_v1_or_v0 & 0x0001) ;
/*************/
/* Compute x */
/*************/
x = ( ( table[0][index0]  + table[0][index1]
      + table[0][index2]  + table[0][index3]
      + table[0][index4]  + table[0][index5]
      + table[0][index6]  + table[0][index7]
      + table[0][index8]  + table[0][index9]
      + table[0][index10] + table[0][index11]
      + table[0][index12] + table[0][index13]
      + table[0][index14] + table[0][index15]
      + 0x08 ) >> 4 ):                    /*Round*/
/*************/
/* Compute y */
/*************/
y = ( ( table[1][index0]  + table[1][index1]
      + table[1][index2]  + table[1][index3]
      + table[1][index4]  + table[1][index5]
      + table[1][index6]  + table[1][index7]
      + table[1][index8]  + table[1][index9]
      + table[1][index10] + table[1][index11]
      + table[1][index12] + table[1][index13]
      + table[1][index14] + table[1][index15]
      + 0x08 ) >> 4 ) :                   /*Round*/
/*************/
/* Compute z */
/*************/
z = ( ( table[2][index0]  + table[2][index1]
      + table[2][index2]  + table[2][index3]
```

FIG.40B

```
        + table[2][index4]  + table[2][index5]
        + table[2][index6]  + table[2][index7]
        + table[2][index8]  + table[2][index9]
        + table[2][index10] + table[2][index11]
        + table[2][index12] + table[2][index13]
        + table[2][index14] + table[2][index15]
        + 0x08 ) >> 4 ) ;                    /*Round*/
    /**********************/
     * Write back results */
    /**********************/
    output[0] = (char) x ;
    output[1] = (char) y ;
    output[2] = (char) z ;
}
```

FIG.40C

```
#define f(v, m, N)  ( v | ( N & ~m ) )
/*************************************************/
/** nonsym_radial_interpolation()                **/
/*************************************************/
void nonsym_radial_interpolation( unsigned char input[],
 unsigned char point[][8], unsigned char resolution[],
 unsigned char output[] )
{
 register unsigned char Amask, Bmask, Cmask ;
 register              int  a, b, c, x, y, z ;
 register unsigned char m0, m1, m2, m3 ;
 register unsigned char v0, v1, v2, v3 ;
 register unsigned char f_v1_v0, f_v2_v1 ;
 register unsigned char f_v2_f_v1_v0, f_v2_v0 ;
 register unsigned char f_v3_v2, f_v3_f_v2_v1 ;
 register unsigned char f_v3_f_v2_f_v1_v0 ;
 register unsigned char f_v3_f_v2_vU, f_v3_v1 ;
 resister unsigned char f_v3_f_v1_v0, f_v3_v0 ;
   /******************/
   /* Generate Masks */
   /******************/
   Amask = ( 1 << resolution[0] ) - 1 ;
   Bmask = ( 1 << resolution[1] ) - 1 ;
   Cmask = ( 1 << resolution[2] ) - 1 ;
   /*************************************/
   /* Snap last value to last table value */
   /*************************************/
   a = ( input[0] == 0x0ff ) ? 0x100 : (int) input[0] ;
   b = ( input[1] == 0x0ff ) ? 0x100 : (int) input[1] ;
   c = ( input[2] == 0x0ff ) ? 0x100 : (int) input[2] ;
 /*************************************************/
   /* Compute slices across masks and input components */
 /*************************************************/
 m0 = ( (Amask & 0x01) << 2 )
    + ( (Bmask & 0x01) << 1 ) + (Cmask & 0x01);
 m1 = ( (Amask & 0x02) << 1 )
    +   (Bmask & 0x02) + ( (Cmask & 0x02) >> 1 );
 m2 =   (Amask & 0x04)
     + ( (Bmask & 0x04) >> 1 ) + ( (Cmask & 0x04) >> 2);
 m3 = ( (Amask & 0x08) >> 1 )
     + ( (Bmask & 0x08) >> 2 ) + ( (Cmask & 0x08) >> 3);
```

```
v0 = m0 & ( ( (a & 0x01) << 2 )
   + ( (b & 0x01) << 1 ) +   (c & 0x01) );
v1 = m1 & ( ( (a & 0x02) << 1 )
   +   (b & 0x02) +       ( (c & 0x02) >> 1 ) ) ;
v2 = m2 & (   (a & 0x04)
   + ( (b & 0x04) >> 1 ) + ( (c & 0x04) >> 2 ) ) ;
v3 = m3 & ( ( (a & 0x08) >> 1 )
   + ( (b & 0x08) >> 2 ) + ( (c & 0x08) >> 3 ) ) ;
/*********************************/
/* Compute offset from origin. */
/*********************************/
f_v1_v0                 = f(v1,m1,v0) ;
f_v2_v1                 = f(v2,m2,v1) ;
f_v2_f_v1_v0            = f(v2,m2,f_v1_v0) ;
f_v2_vU                 = f(v2,m2,vU) ;
f_v3_v2                 = f(v3,m3,v2) ;
f_v3_f_v2_v1            = f(v3,m3,f_v2_v1) ;
f_v3_f_v2_f_v1_v0       = f(v3,m3,f_v2_f_v1_v0) ;
f_v3_f_v2_vU            = f(v3,m3,f_v2_vU) ;
f_v3_vI                 = f(v3,m3,vI) ;
f_v3_f_v1_v0            = f(v3,m3,f_v1_v0) ;
f_v3_vU                 = f(v3,m3,vU) ;
/*************/
/* Compute x */
/*************/
x = ( ( point[0][f_v3_f_v2_f_v1_v0]
      + point[0][f_v2_f_v1_vU]
      + point[0][f_v3_f_v1_v0]
      + point[0][f_v1_vU]
      + point[0][f_v3_f_v2_v0]
      + point[0][f_v2_vU]
      + point[0][f_v3_v0]
      + point[0][vU]
      + point[0][f_v3_f_v2_v1]
      + point[0][f_v2_vI]
```

FIG.41A

DE 698 20 579 T2 2004.10.28

```
    + point[0][f_v3_v1]
    + point[0][v1]
    + point[0][f_v3_v2]
    + point[0][v2]
    + point[0][v3]
    + point[0][0]
    + 0x08 ) >> 4 ) ;
/*****************/
/* Compute y */
/*****************/
y = ( ( point[1][f_v3_f_v2_f_v1_v0]
      + point[1][f_v2_f_v1_v0]
      + point[1][f_v3_f_v1_v0]
      + point[1][f_v1_v0]
      + point[1][f_v3_f_v2_v0]
      + point[1][f_v2_v0]
      + point[1][f_v3_v0]
      + point[1][v0]
      + point[1][f_v3_f_v2_v1]
      + point[1][f_v2_v1]
      + point[1][f_v3_v1]
      + point[1][v1]
      + point[1][f_v3_v2]
      + point[1][v2]
      + point[1][v3]
      + point[1][0]
      + 0x08 ) >> 4 ) ;
/*****************/
/* Compute z */
/*****************/
z = ( ( point[2][f_v3_f_v2_f_v1_v0]
      + point[2][f_v2_f_v1_v0]
      + point[2][f_v3_f_v1_v0]
      + point[2][f_v1_v0]
      + point[2][f_v3_f_v2_v0]
      + point[2][f_v2_v0]
      + point[2][f_v3_v0]
      + point[2][v0]
      + point[2][f_v3_f_v2_v1]
      + point[2][f_v2_v1]
      + point[2][f_v3_v1]
      + point[2][v1]
      + point[2][f_v3_v2]
      + point[2][v2]
      + point[2][v3]
      + point[2][0]
      + 0x08 ) >> 4 ) ;
/*****************/
/* Write back results */
/*****************/
output[0] = (char) x ;
output[1] = (char) y ;
output[2] = (char) z ;
}
```

/*Round*/

/*Round*/

/*Round*/

FIG.41B

```
-------------------------------------------------
-- Pruned Tetrahedral Interpolation             --
-------------------------------------------------
entity pruned_tetrahedral is
port (
     -- Interpolation Select --
     tetra_not_radial: in  INPUT_ARRAY ;
     -- Input Pixel --
     input:            in  INPUT_ARRAY ;
     -- Vertex Points of Cube from Color Table --
     point:            in  POINT_ARRAY ;
     -- Output Pixel --
     output:           out OUTPUT_ARRAY
) ;
end pruned_tetrahedral ;
-------------------------------------------------
architecture pruned_tetrtl of pruned_tetrahedral is
-------------------------------------------------
 constant VERTEX_COUNT: INTEGER := 2**NUM_INTERP_BITS;
 signal   vertice:      VERTEX_ARRAY ;
 signal   point_value:  POINT_VALUE_ARRAY ;
begin
-------------------------------------------------
-- Compute vertices                            --
-------------------------------------------------
VERTICIES: process ( tetra_not_radial, input )
     variable vertex:    STD_LOGIC_VECTOR(2 downto 0) ;
     variable temp1, temp2:      VERTEX_ARRAY ;
     variable limit, prev_limit:  INTEGER ;
begin
     ---------------------
     -- Initialize Array --
     ---------------------
     limit := 0 ;
     temp2 := ( others => "000" ) ;
     -----------------------------------------
     -- For each slice across the interpolate bits --
     -----------------------------------------
     for i in 0 to NUM_INTERP_BITS-1 loop
          ---------------------------------
          -- Load slice into limit location --
```

```
          ---------------------------------
          vertex := input(0)(i) & input(1)(i)
                                & input(2)(i) ;
          ---------------------------------
          -- Select Radial or Tetrahedral --
          ---------------------------------
          if ( tetra_not_radial = '0' ) then
               -----------
               -- Radial --
               -----------
               ------------------------
               -- limit is 1, 2, 4, 8 --
               ------------------------
               prev_limit := limit ;
               limit := 2**i ;
               ---------------------------
               -- Assign multiplex selects --
               ---------------------------
               for j in (VERTEX_COUNT/2)-1 downto 0
               loop
                    if ( j < limit ) then
                         temp2(prev_limit + j) := vertex
                    else
                         temp2 := temp2 ;
                    end if ;
               end loop ;
          else
               ---------------
               -- Tetrahedral --
               ---------------
               ---------------------------------
               -- limit is 0, 2, 6, 14, 30, ... --
               ---------------------------------
               limit := ( 2**(i+1) ) - 2 ;
               ---------------------------------
               -- load slice into limit location --
```

FIG.42A

```
                    ----------------------------------
                    temp2(limit) := vertex ;
                    temp1 := temp2 ;
                    ----------------------------
                    -- for all values in temp1 --
                    ----------------------------
                    for j in (VERTEX_COUNT/2)-1 downto 0
                    loop
                         if ( j < limit/2 ) then
                              temp2(2*j)   := temp1(limit)
                                              and temp1(j) ;
                            temp2((2*j)+1) := temp1(limit)
                                              or  temp1(j) ;
                         else
                              temp2 := temp2 ;
                         end if ;
                    end loop ;
               end if ;
          end loop ;
          -------------------------
          -- Move into index array --
          -------------------------
          vertice <= temp2 ;
end process ;
-------------------------------------------------
-- Select the values of vertices to be used by the --
-- interpolation. The last two vertices are always --
-- point(0) and the rounding constant.             --
-------------------------------------------------
MUX: process ( vertice, point )
begin
     for d in 0 to NUM_OUTPUT_DIMEN-1 loop
        point_value(d) <= (others => (others => '0'));
        for i in 0 to VERTEX_COUNT-2 loop
          case vertice(i) is
          when "000"  => point_value(d)(i)(7 downto 0)
                                          <= point(d)(0) ;
          when "001"  => point_value(d)(i)(7 downto 0)
                                          <= point(d)(1) ;
          when "010"  => point_value(d)(i)(7 downto 0)
                                          <= point(d)(2) ;
          when "011"  => point_value(d)(i)(7 downto 0)
                                          <= point(d)(3) ;
          when "100"  => point_value(d)(i)(7 downto 0)
                                          <= point(d)(4) ;
          when "101"  => point_value(d)(i)(7 downto 0)
                                          <= point(d)(5) ;
          when "110"  => point_value(d)(i)(7 downto 0)
                                          <= point(d)(6) ;
          when others => point_value(d)(i)(7 downto 0)
                                          <= point(d)(7) ;
          end case ;
        end loop ;
        point_value(d)(VERTEX_COUNT-1)
                        (7 downto 0) <= point(d)(0) ;
        point_value(d)(VERTEX_COUNT)
                        (NUM_INTERP_BITS-1) <= '1' ;
     end loop ;
end process ;
-------------------------------------------------
-- SUM adds the selected vertices values and   --
-- truncates to eight bits. The process is     --
-- implemented to minimize prop delay by       --
-- successive pairing and adding.              --
-------------------------------------------------
SUM: process ( point_value )
     variable  temp:      POINT_VALUES ;
     variable  bound:     INTEGER ;
begin
     for d in 0 to NUM_OUTPUT_DIMEN-1 loop
        temp := point_value(d) ;
        bound := VERTEX_COUNT + 1 ;
        for i in 0 to NUM_INTERP_BITS loop
             for j in 0 to VERTEX_COUNT/2 loop
                  if ( 2*j < bound ) then
                       temp(j) := temp(2*j) ;
                  else
                       temp(j) := (others => '0') ;
                  end if ;
                  if ( (2*j)+1 < bound ) then
```

FIG.42B

```
            temp(j) := temp(j) + temp((2*j)+1) ;
        else
            temp(j) := temp(j) ;
        end if ;
    end loop ;
    bound := ( bound / 2 ) + ( bound mod 2 ) ;
end loop ;
output(d) <= temp(0)(7+NUM_INTERP_BITS
                downto NUM_INTERP_BITS) ;
end loop ;
end process ;
--------------------------------------
end pruned_tetrtl ;
--------------------------------------
```

FIG.42C

```
-------------------------------------------------
-- Pruned Tetrahedral Interpolation             --
-------------------------------------------------
entity pruned_tetrahedral is
port (
     -- Input Pixel --
     input:          in    INPUT_ARRAY ;
     -- Vertex Points of Cube from Color Table --
     point:          in    POINT_ARRAY ;
     -- Output Pixel --
     output:         out   OUTPUT_ARRAY
) ;
end pruned_tetrahedral ;
-------------------------------------------------
architecture pruned_tetrtl of pruned_tetrahedral is
-------------------- ----------------------------
 constant VERTEX_COUNT: INTEGER := 2**NUM_INTERP_BITS;
 signal   vertex:       VERTEX_ARRAY ;
 signal   point_value:  POINT_VALUE_ARRAY ;
begin
-------------------------------------------------
-- Compute vertices.                           --
-------------------------------------------------
VERTICIES: process ( input )
     variable  temp1, temp2:   VERTEX_ARRAY ;
     variable  limit:          INTEGER ;
begin
     --------------------
     -- Initialize Array --
     --------------------
     temp2 := ( others => "000" ) ;
     ------------------------------------------
     -- For each slice across the interpolate bits --
     ------------------------------------------
     for i in 0 to NUM_INTERP_BITS-1 loop
          ---------------------------------
          -- limit is 0, 2, 6, 14, 30, ... --
          ---------------------------------
          limit := ( 2**(i+1) ) - 2 ;
          ---------------------------------
          -- Load slice into limit location --
          ---------------------------------
          temp2(limit) := input(0)(i) & input(1)(i)
                                      & input(2)(i);
          temp1 := temp2 ;
          ---------------------------
          -- for all values in temp1 --
          ---------------------------
          for j in (VERTEX_COUNT/2)-1 downto 0 loop
               if ( j < limit/2 ) then
                    temp2(2*j)     := temp1(limit)
                                   and temp1(j) ;
                    temp2((2*j)+1) := temp1(limit)
                                   or  temp1(j) ;
               else
                    temp2 := temp2 ;
               end if ;
          end loop ;
     end loop ;
     ------------------------
     -- Move into index array --
     ------------------------
     vertex <= temp2 ;
end process ;

-------------------------------------------------
-- Select the values of vertices to be used by the --
-- interpolation. The last two vertices are always --
-- point(0) and the rounding constant.          --
-------------------------------------------------
MUX: process ( vertex, point )
begin
     ----------------------------
     -- For each output dimension --
     ----------------------------
     for d in 0 to NUM_OUTPUT_DIMEN-1 loop
        --------------
        -- Initialize --
        --------------
        point_value(d) <= (others => (others => '0' ));
```

FIG.43A

DE 698 20 579 T2 2004.10.28

```
        ----------------------------------------
        -- point_value(d)(0 : VERTEX_COUNT-2) --
        ----------------------------------------
        for i in 0 to VERTEX_COUNT-2 loop
          case vertex(i) is
          when "000"  => point_value(d)(i)(7downto 0)
                              <= point(d)(0) ;
          when "001"  => point_value(d)(i)(7 downto 0)
                              <= point(d)(1) ;
          when "010"  => point_value(d)(i)(7 downto 0)
                              <= point(d)(2) ;
          when "011"  => point_value(d)(i)(7 downto 0)
                              <= point(d)(3) ;
          when "100"  => point_value(d)(i)(7 downto 0)
                              <= point(d)(4) ;
          when "101"  => point_value(d)(i)(7 downto 0)
                              <= point(d)(5) ;
          when "110"  => point_value(d)(i)(7 downto 0)
                              <= point(d)(6) ;
          when others => point_value(d)(i)(7 downto 0)
                              <= point(d)(7) ;
          end case ;
        end loop ;
        ----------------
        -- point(d)(0) --
        ----------------
        point_value(d)(VERTEX_COUNT-1)
                        (7 downto 0) <= point(d)(0) ;
        ----------------------
        -- Rounding constant --
        ----------------------
        point_value(d)(VERTEX_COUNT)
                        (NUM_INTERP_BITS-1) <= '1' ;
    end loop ;
end process ;
------------------------------------------------
-- SUM adds the selected vertices values and  --
-- truncates to eight bits. The process is    --
-- implemented to minimize prop delay by      --
-- successive pairing and adding.             --
------------------------------------------------
SUM: process ( point_value )
    variable  temp:     POINT_VALUES ;
    variable  bound:    INTEGER ;
begin
    ----------------------------
```

```
-- For each output dimension --
-------------------------------
for d in 0 to NUM_OUTPUT_DIMEN-1 loop
   -------------------------------
   -- Initialize temp and bound. --
   -------------------------------
   temp := point_value(d) ;
   bound := VERTEX_COUNT + 1 ;
   ---------------------------------------
   -- Loop for ceil(log2(VERTEX_COUNT)) --
   ---------------------------------------
   for i in 0 to NUM_INTERP_BITS loop
        ------------------------------------
        -- Pair temp array elements and add. --
        ------------------------------------
        for j in 0 to VERTEX_COUNT/2 loop
             ----------------------------
             -- Step 1: Move temp(even's) --
             ----------------------------
             if ( 2*j < bound ) then
                  temp(j) := temp(2*j) ;
             else
                  temp(j) := (others => '0') ;
             end if ;
             ----------------------------
             -- Step 2: Add temp(odd's) --
             ----------------------------
             if ( (2*j)+1 < bound ) then
                  temp(j) := temp(j) + temp((2*j)+1)
             else
                  temp(j) := temp(j) ;
             end if ;
```

FIG.43B

```
        end loop ;
        ------------------------------
        -- Set bound for next iteration --
        ------------------------------
            bound := ( bound / 2 ) + ( bound mod 2 ) ;
        end loop ;
        ------------------------
        -- Truncate output. --
        ------------------------
        output(d) <= temp(0)(7+NUM_INTERP_BITS
                        downto NUM_INTERP_BITS) ;
    end loop ;
end process ;
------------------------------------------
end pruned_tetrtl ;
------------------------------------------
```

FIG.43C

```
-----------------------------------------
-- Non-Symmetric Radial Interpolation --
-----------------------------------------
entity nonsym_radial is
port (
     -- Input Pixel --
     input:          in    INPUT_ARRAY ;
     -- Vertex Points of Cube from Color Table --
     point:          in    POINT_ARRAY ;
     -- Lattice Resolution --
     resolution:     in    RESOLUTION_ARRAY ;
     -- Output Pixel --
     output:         out   OUTPUT_ARRAY
) ;
end nonsym_radial ;
-------------------------------------------------
architecture nonsym_radialrtl of nonsym_radial is
-------------------- ----------------------------
 constant VERTEX_COUNT: INTEGER := 2**NUM_INTERP_BITS;
 signal   mask:         MASK_ARRAY ;
 signal   vertex:       VERTEX_ARRAY ;
 signal   point_value:  POINT_VALUE_ARRAY ;
begin
-------------------------------------------------
-- Compute Lattice Resolution Masks             --
-------------------------------------------------
MASKS: process ( resolution )
     variable  top:           INTEGER ;
begin
     ----------------------
     -- For each Component --
     ----------------------
     for i in 0 to 2 loop
       top := 0 ;
       --------------------------------
       -- Convert BIT_VECTOR to INTEGER --
       --------------------------------
       for j in 2 downto 0 loop
            if ( resolution(i)(j) = '1' ) then
                 top := top + 2**j ;
            else
                 top := top ;
            end if ;
       end loop ;
       -----------------
       -- Generate Mask --
       -----------------
       for j in NUM_INTERP_BITS-1 downto 0 loop
            if ( j < top ) then
                 mask(j)(i) <= '1' ;
            else
                 mask(j)(i) <= '0' ;
            end if ;
       end loop ;
     end loop ;
end process ;
-------------------------------------------------
-- Compute vertices.                           --
-------------------------------------------------
VERTICIES: process ( input, mask )
     variable  temp1, temp2:   VERTEX_ARRAY ;
     variable  limit:          INTEGER ;
begin
     ---------------------
     -- Initialize Arrays --
     ---------------------
     temp2 := ( others => "000" ) ;
     -----------------------------------------
     -- For each slice across the interpolate bits --
     -----------------------------------------
     for i in 0 to NUM_INTERP_BITS-1 loop
          ---------------------------------
          -- limit is 0, 2, 6, 14, 30, ... --
          ---------------------------------
          limit := ( 2**(i+1) ) - 2 ;
          ---------------------------------
          -- Load slice into limit location --
```

FIG.44A

```
------------------------------------
temp2(limit) := mask(i) and ( input(0)(i)
              & input(1)(i) & input(2)(i) ) ;
temp1 := temp2 ;
---------------------------
-- for all values in temp1 --
---------------------------
for j in (VERTEX_COUNT/2)-1 downto 0 loop
    if ( j < limit/2 ) then
        -------------------
        -- P'[2*j] = P[j] --
        -------------------
        temp2(2*j)      := temp1(j) ;
        ------------------------------
        -- P'[2*j+1]=v[i]|(P[j] & ~m[i])--
        ------------------------------
        temp2((2*j)+1) := temp1(limit)
              or ( temp1(j) and not mask[i] );
    else
        temp2 := temp2 ;
    end if ;
  end loop ;
end loop ;
-------------------------
-- Move into index array --
-------------------------
vertex <= temp2 ;
end process ;
-----------------------------------------------------
-- Select the values of vertices to be used by the --
-- interpolation. The last two vertices are always --
-- point(0) and the rounding constant.             --
-----------------------------------------------------
MUX: process ( vertex, point )
begin
    for d in 0 to NUM_OUTPUT_DIMEN-1 loop
      point_value(d) <= (others => (others => '0'));
      for i in 0 to VERTEX_COUNT-2 loop
        case vertex(i) is
      when "000"  => point_value(d)(i)(7 downto 0)
                                        <= point(d)(0) ;
      when "001"  => point_value(d)(i)(7 downto 0)
                                        <= point(d)(1) ;
      when "010"  => point_value(d)(i)(7 downto 0)
                                        <= point(d)(2) ;
      when "011"  => point_value(d)(i)(7 downto 0)
                                        <= point(d)(3) ;
      when "100"  => point_value(d)(i)(7 downto 0)
                                        <= point(d)(4) ;
      when "101"  => point_value(d)(i)(7 downto 0)
                                        <= point(d)(5) ;
      when "110"  => point_value(d)(i)(7 downto 0)
                                        <= point(d)(6) ;
      when others => point_value(d)(i)(7 downto 0)
                                        <= point(d)(7) ;
        end case ;
      end loop ;
      point_value(d)(VERTEX_COUNT-1)
                    (7 downto 0) <= point(d)(0) ;
      point_value(d)(VERTEX_COUNT)
                    (NUM_INTERP_BITS-1) <= '1' ;
    end loop ;
end process ;
-----------------------------------------------------
-- Select the values of vertices to be used by the --
-- interpolation. The last two vertices are always --
-- point(0) and the rounding constant.             --
-----------------------------------------------------
SUM: process ( point_value )
    variable  temp:      POINT_VALUES ;
    variable  bound:     INTEGER ;
begin
    for d in 0 to NUM_OUTPUT_DIMEN-1 loop
      temp := point_value(d) ;
      bound := VERTEX_COUNT + 1 ;
      for i in 0 to NUM_INTERP_BITS loop
          for j in 0 to VERTEX_COUNT/2 loop
              if ( 2*j < bound ) then
```

FIG.44B

```
            temp(j) := temp(2*j) ;
        else
            temp(j) := (others => '0') ;
        end if ;
        if ( (2*j)+1 < bound ) then
            temp(j) := temp(j) + temp((2*j)+1) ;
        else
            temp(j) := temp(j) ;
        end if ;
    end loop ;
    bound := ( bound / 2 ) + ( bound mod 2 ) ;
end loop ;
output(d) <= temp(0)(7+NUM_INTERP_BITS
                downto NUM_INTERP_BITS) ;
    end loop ;
end process ;

-------------------------------------------
end nonsym_radialrtl ;
-------------------------------------------
```

FIG.44C

```
define g(v, m, N)  ( ( v | ~m ) & N )
define h(v, N)     ( v | N )
*************************************************/
/** nonsym_pruned_tetrahedral()                  **/
/*************************************************/
void nonsym_pruned_tetrahedral( unsigned char input[],
   unsigned char point[][8], unsigned char resolution[],
   unsigned char output[] )

   register unsigned char Amask, Bmask, Cmask ;
   register               int  a, b, c, x, y, z ;
   register unsigned char m0, m1, m2, m3 ;
   register unsigned char v0, v1, v2, v3 ;
   register unsigned char g_v1_v0, h_v1_v0 ;
   register unsigned char g_v2_v1, g_v2_g_v1_v0 ;
   register unsigned char g_v2_h_v1_v0, h_v2_v1 ;
   register unsigned char h_v2_g_v1_v0, h_v2_h_v1_v0 ;
   register unsigned char g_v3_v2, g_v3_g_v2_v1 ;
   register unsigned char g_v3_h_v2_v1 ;
   register unsigned char g_v3_g_v2_g_v1_v0 ;
   register unsigned char g_v3_g_v2_h_v1_v0 ;
   register unsigned char g_v3_h_v2_g_v1_v0 ;
   register unsigned char g_v3_h_v2_h_v1_v0 ;
   register unsigned char h_v3_v2, h_v3_g_v2_v1 ;
   register unsigned char h_v3_h_v2_v1 ;
   register unsigned char h_v3_g_v2_g_v1_v0 ;
   register unsigned char h_v3_g_v2_h_v1_v0 ;
   register unsigned char h_v3_h_v2_g_v1_v0 ;
   register unsigned char h_v3_h_v2_h_v1_v0 ;
   /******************/
   /* Generate Masks */
   /******************/
   Amask = ( 1 << resolution[0] ) - 1 ;
   Bmask = ( 1 << resolution[1] ) - 1 ;
   Cmask = ( 1 << resolution[2] ) - 1 ;
   /*************************************/
   /* Snap last value to last table value */
   /*************************************/
   a = ( input[0] == 0x0ff ) ? 0x100 : (int) input[0] ;
   b = ( input[1] == 0x0ff ) ? 0x100 : (int) input[1] ;
   c = ( input[2] == 0x0ff ) ? 0x100 : (int) input[2] ;
```

```
   /************************************************/
   /*Compute slices across masks and input components */
   /************************************************/
   m0 = ( (Amask & 0x01) << 2 )
      + ( (Bmask & 0x01) << 1 ) + (Cmask & 0x01);
   m1 = ( (Amask & 0x02) << 1 )
      +   (Bmask & 0x02) + ( (Cmask & 0x02) >> 1 );
   m2 =   (Amask & 0x04)
      + ( (Bmask & 0x04) >> 1 ) + ( (Cmask & 0x04) >> 2);
   m3 = ( (Amask & 0x08) >> 1 )
      + ( (Bmask & 0x08) >> 2 ) + ( (Cmask & 0x08) >> 3 );
   v0 = m0 & ( ( (a & 0x01) << 2 )
      + ( (b & 0x01) << 1 ) +   (c & 0x01) ) ;
   v1 = m1 & ( ( (a & 0x02) << 1 )
      +   (b & 0x02) +        ( (c & 0x02) >> 1 ) ) ;
   v2 = m2 & (   (a & 0x04)
      + ( (b & 0x04) >> 1 ) + ( (c & 0x04) >> 2 ) ) ;
   v3 = m3 & ( ( (a & 0x08) >> 1 )
      + ( (b & 0x08) >> 2 ) + ( (c & 0x08) >> 3 ) ) ;
   /*******************************/
   /* Compute offset from origin. */
   /*******************************/
   g_v1_v0                = g(v1,m1,v0) ;
   h_v1_v0                = h(v1,v0) ;
   g_v2_v1                = g(v2,m2,v1) ;
   g_v2_g_v1_v0           = g(v2,m2,g_v1_v0) ;
   g_v2_h_v1_v0           = g(v2,m2,h_v1_v0) ;
   h_v2_v1                = h(v2,v1) ;
   h_v2_g_v1_v0           = h(v2,g_v1_v0) ;
   h_v2_h_v1_v0           = h(v2,h_v1_v0) ;
   g_v3_v2                = g(v3,m3,v2) ;
   g_v3_g_v2_v1           = g(v3,m3,g_v2_v1) ;
   g_v3_g_v2_g_v1_v0      = g(v3,m3,g_v2_g_v1_v0) ;
   g_v3_g_v2_h_v1_v0      = g(v3,m3,g_v2_h_v1_v0) ;
   g_v3_h_v2_v1           = g(v3,m3,h_v2_v1) ;
```

FIG.45A

```
g_v3_h_v2_g_v1_v0      = g(v3,m3,h_v2_g_v1_v0) ;
g_v3_h_v2_h_v1_v0      = g(v3,m3,h_v2_h_v1_v0) ;
h_v3_v2                = h(v3,v2) ;
h_v3_g_v2_v1           = h(v3,g_v2_v1) ;
h_v3_g_v2_g_v1_v0      = h(v3,g_v2_g_v1_v0) ;
h_v3_g_v2_h_v1_v0      = h(v3,g_v2_h_v1_v0) ;
h_v3_h_v2_vI           = h(v3,h_v2_vI) ;
h_v3_h_v2_g_v1_v0      = h(v3,h_v2_g_v1_v0) ;
h_v3_h_v2_h_v1_v0      = h(v3,h_v2_h_v1_v0) ;
/*************/
/* Compute x */
/*************/
x = ( ( point[0][0]
      + point[0][v3]
      + point[0][g_v3_v2]
      + point[0][g_v3_g_v2_v1]
      + point[0][g_v3_g_v2_g_v1_v0]
      + point[0][g_v3_g_v2_h_v1_v0]
      + point[0][g_v3_h_v2_vI]
      + point[0][g_v3_h_v2_g_v1_v0]
      + point[0][g_v3_h_v2_h_v1_v0]
      + point[0][h_v3_v2]
      + point[0][h_v3_g_v2_v1]
      + point[0][h_v3_g_v2_g_v1_v0]
      + point[0][h_v3_g_v2_h_v1_v0]
      + point[0][h_v3_h_v2_vI]
      + point[0][h_v3_h_v2_g_v1_v0]
      + point[0][h_v3_h_v2_h_v1_v0]
      + 0x08 ) >> 4 ) ;                    /*Round*/
/*************/
/* Compute y */
/*************/
y = ( ( point[1][0]
      + point[1][v3]
      + point[1][g_v3_v2]
      + point[1][g_v3_g_v2_v1]
      + point[1][g_v3_g_v2_g_v1_v0]
      + point[1][g_v3_g_v2_h_v1_v0]
      + point[1][g_v3_h_v2_vI]
      + point[1][g_v3_h_v2_g_v1_v0]
      + point[1][g_v3_h_v2_h_v1_v0]
      + point[1][h_v3_v2]
      + point[1][h_v3_g_v2_v1]
      + point[1][h_v3_g_v2_g_v1_v0]
      + point[1][h_v3_g_v2_h_v1_v0]
      + point[1][h_v3_h_v2_vI]
      + point[1][h_v3_h_v2_g_v1_v0]
      + point[1][h_v3_h_v2_h_v1_v0]
      + 0x08 ) >> 4 ) ;                    /*Round*/
/*************/
/* Compute z */
/*************/
z = ( ( point[2][0]
      + point[2][v3]
      + point[2][g_v3_v2]
      + point[2][g_v3_g_v2_v1]
      + point[2][g_v3_g_v2_g_v1_v0]
      + point[2][g_v3_g_v2_h_v1_v0]
      + point[2][g_v3_h_v2_vI]
      + point[2][g_v3_h_v2_g_v1_v0]
      + point[2][g_v3_h_v2_h_v1_v0]
      + point[2][h_v3_v2]
      + point[2][h_v3_g_v2_v1]
      + point[2][h_v3_g_v2_g_v1_v0]
      + point[2][h_v3_g_v2_h_v1_v0]
      + point[2][h_v3_h_v2_vI]
      + point[2][h_v3_h_v2_g_v1_v0]
      + point[2][h_v3_h_v2_h_v1_v0]
      + 0x08 ) >> 4 ) ;                    /*Round*/

/*********************/
/* Write back results */
/*********************/
output[0] = (char) x ;
output[1] = (char) y ;
output[2] = (char) z ;

}
```

FIG.45B

```
-------------------------------------------------
-- Non-Symmetric Pruned Tetrahedral Interpolation --
-------------------------------------------------
entity nonsym_pruned_tetrahedral is
port (
     -- Input Pixel --
     input:          in    INPUT_ARRAY ;
     -- Vertex Points of Cube from Color Table --
     point:          in    POINT_ARRAY ;
     -- Lattice Resolution --
     resolution:     in    RESOLUTION_ARRAY ;
     -- Output Pixel --
     output:         out   OUTPUT_ARRAY
) ;
end nonsym_pruned_tetrahedral ;
-------------------------------------------------
architecture nonsym_tetrtl of nonsym_pruned_tetrahedral is
-------------------------------------------------
 constant VERTEX_COUNT: INTEGER := 2**NUM_INTERP_BITS;
 signal   mask:         MASK_ARRAY ;
 signal   vertex:       VERTEX_ARRAY ;
 signal   point_value:  POINT_VALUE_ARRAY ;
begin
-------------------------------------------------
-- Compute Lattice Resolution Masks             --
-------------------------------------------------
MASKS: process ( resolution )
    variable  top:           INTEGER ;
begin
    ------------------------
    -- For each Component --
    ------------------------
    for i in 0 to 2 loop
      top := 0 ;
      ---------------------------------
      -- Convert BIT_VECTOR to INTEGER --
      ---------------------------------
      for j in 2 downto 0 loop
          if ( resolution(i)(j) = '1' ) then
               top := top + 2**j ;
          else
               top := top ;
          end if ;
      end loop ;
      ------------------
      -- Generate Mask --
      ------------------
      for j in NUM_INTERP_BITS-1 downto 0 loop
          if ( j <= top ) then
               mask(j)(i) <= '1' ;
          else
               mask(j)(i) <= '0' ;
          end if ;
      end loop ;
    end loop ;
end process ;
-------------------------------------------------
-- Compute vertices.                           --
-------------------------------------------------
VERTICIES: process ( input, mask )
    variable  temp1, temp2:   VERTEX_ARRAY ;
    variable  limit:          INTEGER ;
begin
    ----------------------
    -- Initialize Arrays --
    ----------------------
    temp2 := ( others => "000" ) ;
    -------------------------------------------------
    -- For each slice across the interpolate bits --
    -------------------------------------------------
    for i in 0 to NUM_INTERP_BITS-1 loop
        ---------------------------------
        -- limit is 0, 2, 6, 14, 30, ... --
        ---------------------------------
        limit := ( 2**(i+1) ) - 2 ;
        ---------------------------------
```

FIG.46A

```
        -- Load slice into limit location
        ----------------------------------
        temp2(limit) := mask(i) and ( input(0)(i)
                      & input(1)(i) & input(2)(i) ) ;
        templ := temp2 ;
        ---------------------------
        -- for all values in templ --
        ---------------------------
        for j in (VERTEX_COUNT/2)-1 downto 0 loop
            if ( j < limit/2 ) then
        ---------------------------------
        --      -- P'[2*j]=(v[i] | ~m[i]) & P[j]--
        ---------------------------------
                temp2(2*j)   := ( templ(limit)
                              or not mask(i) ) and templ(j) ;
                -------------------------
                -- P'[2*j+1] = v[i] | P[j] --
                -------------------------
                temp2((2*j)+1) := templ(limit)
                                  or templ(j) ;
            else
                temp2 := temp2 ;
            end if ;
        end loop ;
    end loop ;
    -------------------------
    -- Move into index array --
    ------- ----------------
    vertex <= temp2 ;
end process ;
-------------------------------------------------
-- Select the values of vertices to be used by the --
-- interpolation. The last two vertices are always --
-- point(0) and the rounding constant.             --
-------------------------------------------------
MUX: process ( vertex, point )
begin
    for d in 0 to NUM_OUTPUT_DIMEN-1 loop
      point_value(d) <= (others => (others => '0'));
      for i in 0 to VERTEX_COUNT-2 loop
        case vertex(i) is
      when "000"  => point_value(d)(i)(7 downto 0)
                                  <= point(d)(0) ;
      when "001"  => point_value(d)(i)(7 downto 0)
                                  <= point(d)(1) ;
      when "010"  => point_value(d)(i)(7 downto 0)
                                  <= point(d)(2) ;
      when "011"  => point_value(d)(i)(7 downto 0)
                                  <= point(d)(3) ;
      when "100"  => point_value(d)(i)(7 downto 0)
                                  <= point(d)(4) ;
      when "101"  => point_value(d)(i)(7 downto 0)
                                  <= point(d)(5) ;
      when "110"  => point_value(d)(i)(7 downto 0)
                                  <= point(d)(6) ;
      when others => point_value(d)(i)(7 downto 0)
                                  <= point(d)(7) ;
        end case ;
      end loop ;
      point_value(d)(VERTEX_COUNT-1)
                          (7 downto 0) <= point(d)(0) ;
      point_value(d)(VERTEX_COUNT)
                          (NUM_INTERP_BITS-1) <= '1' ;
    end loop ;
end process ;

-------------------------------------------------
-- Select the values of vertices to be used by the --
-- interpolation. The last two vertices are always --
-- point(0) and the rounding constant.             --
-------------------------------------------------
SUM: process ( point_value )
    variable  temp:     POINT_VALUES ;
    variable  bound:    INTEGER ;
begin
    for d in 0 to NUM_OUTPUT_DIMEN-1 loop
      temp := point_value(d) ;
      bound := VERTEX_COUNT + 1 ;
      for i in 0 to NUM_INTERP_BITS loop
          for j in 0 to VERTEX_COUNT/2 loop
```

FIG.46B

```
            if ( 2*j < bound ) then
                temp(j) := temp(2*j) ;
            else
                temp(j) := (others => '0') ;
            end if ;
            if ( (2*j)+1 < bound ) then
                temp(j) := temp(j) + temp((2*j)+1) ;
            else
                temp(j) := temp(j) ;
            end if ;
        end loop ;
        bound := ( bound / 2 ) + ( bound mod 2 ) ;
    end loop ;
    output(d) <= temp(0)(7+NUM_INTERP_BITS
                    downto NUM_INTERP_BITS) ;
  end loop ;
end process ;

-------------------------------------------
end nonsym_pruned_tetrtl ;
-------------------------------------------
```

FIG.46C

```
-------------------------------------------------
-- Non-Symmetric Pruned Tetrahedral Interpolation --
-------------------------------------------------
entity nonsym_radial_tetr is
port (
     -- Interpolation Select ---
     tetra_not_radial: in  STD_LOGIC ;
     -- Input Pixel --
     input:            in  INPUT_ARRAY ;
     -- Vertex Points of Cube from Color Table --
     point:            in  POINT_ARRAY ;
     -- Lattice Resolution --
     resolution:       in  RESOLUTION_ARRAY ;
     -- Output Pixel --
     output:           out OUTPUT_ARRAY
) ;
end nonsym_pruned_tetrahedral ;
-------------------------------------------------
architecture nonsym_radial_tetrtl of nonsym_radial_tetr is
-------------------- ----------------------------
 constant VERTEX_COUNT: INTEGER := 2**NUM_INTERP_BITS;
 signal   mask:         MASK_ARRAY ;
 signal   vertex:       VERTEX_ARRAY ;
 signal   point_value:  POINT_VALUE_ARRAY ;
begin
-------------------------------------------------
-- Compute Lattice Resolution Masks             --
-------------------------------------------------
MASKS: process ( resolution )
    variable  top:           INTEGER ;
begin
    for i in 0 to 2 loop
      top := 0 ;
      for j in 2 downto 0 loop
          if ( resolution(i)(j) = '1' ) then
              top := top + 2**j ;
          else
              top := top ;
          end if ;
      end loop ;
      for j in NUM_INTERP_BITS-1 downto 0 loop
          if ( j < top ) then
              mask(j)(i) <= '1' ;
          else
              mask(j)(i) <= '0' ;
          end if ;
      end loop ;
    end loop ;
end process ;
-------------------------------------------------
-- Compute vertices.                            --
-------------------------------------------------
VERTICIES: process ( tetra_not_radial, input, mask )
    variable  temp1, temp2:   VERTEX_ARRAY ;
    variable  limit:          INTEGER ;
begin
    temp2 := ( others => "000" ) ;
    for i in 0 to NUM_INTERP_BITS-1 loop
        limit := ( 2**(i+1) ) - 2 ;
        temp2(limit) := mask(i) and ( input(0)(i)
                      & input(1)(i) & input(2)(i) ) ;
        temp1 := temp2 ;
        for j in (VERTEX_COUNT/2)-1 downto 0 loop
            if ( j < limit/2 ) then
                ------------------------------
                -- Select Radial or Tetrahedral--
                ------------------------------
                if ( tetra_not_radial = '0' ) then
                    -----------
                    -- Radial --
                    -----------
                    ------------------
                    -- P'[2*j] = P[j] --
                    ------------------
                    temp2(2*j)     := temp1(j) ;
                    ------------------------------
                    -- P'[2*j+1]=v[i]|(P[j]&~m[i])
```

FIG.47A

```
                    ----------------------------
                    temp2((2*j)+1) := temp1(limit)
                    or (temp1(j) and not mask(i));
                else
                    ----------------------------
                    -- NonSym Pruned Tetrahedral -
                    ----------------------------
                    ----------------------------
                    -- P'[2*j]=(v[i]|~m[i])&P[j]--
                    ----------------------------
                    temp2(2*j)    := ( temp1(limit)
                    or not mask(i) ) and temp1(j) ;
                    ----------------------------
                    -- P'[2*j+1] = v[i]|P[j] --
                    ----------------------------
                    temp2((2*j)+1) := temp1(limit)
                     or temp1(j) ;
                end if ;
            else
                temp2 := temp2 ;
            end if ;
        end loop ;
    end loop ;
    vertex <= temp2 ;
end process ;
-------------------------------------------------
-- Select the values of vertices to be used by the --
-- interpolation. The last two vertices are always --
-- point(0) and the rounding constant.             --
-------------------------------------------------
MUX: process ( vertex, point )
begin
    for d in 0 to NUM_OUTPUT_DIMEN-1 loop
      point_value(d) <= (others => (others => '0'));
      for i in 0 to VERTEX_COUNT-2 loop
        case vertex(i) is
      when "000"  => point_value(d)(i)(7 downto 0)
                                    <= point(d)(0) ;
      when "001"  => point_value(d)(i)(7 downto 0)
                                    <= point(d)(1) ;
      when "010"  => point_value(d)(i)(7 downto 0)
                                    <= point(d)(2)
      when "011"  => point_value(d)(i)(7 downto 0)
                                    <= point(d)(3)
      when "100"  => point_value(d)(i)(7 downto 0)
                                    <= point(d)(4)
      when "101"  => point_value(d)(i)(7 downto 0)
                                    <= point(d)(5)
      when "110"  => point_value(d)(i)(7 downto 0)
                                    <= point(d)(6) ;
      when others => point_value(d)(i)(7 downto 0)
                                    <= point(d)(7) ;
        end case ;
      end loop ;
      point_value(d)(VERTEX_COUNT-1)
                    (7 downto 0) <= point(d)(0) ;
      point_value(d)(VERTEX_COUNT)
                    (NUM_INTERP_BITS-1) <= '1' ;
    end loop ;
end process ;
-------------------------------------------------
-- Select the values of vertices to be used by the --
-- interpolation. The last two vertices are always --
-- point(0) and the rounding constant.             --
-------------------------------------------------
SUM: process ( point_value )
    variable  temp:     POINT_VALUES ;
    variable  bound:    INTEGER ;
begin
    for d in 0 to NUM_OUTPUT_DIMEN-1 loop
      temp := point_value(d) ;
      bound := VERTEX_COUNT + 1 ;
      for i in 0 to NUM_INTERP_BITS loop
          for j in 0 to VERTEX_COUNT/2 loop
              if ( 2*j < bound ) then
                  temp(j) := temp(2*j) ;
              else
                  temp(j) := (others => '0') ;
```

FIG.47B

```
                    end if ;
                    if ( (2*j)+1 < bound ) then
                         temp(j) := temp(j) + temp((2*j)+1)
                    else
                         temp(j) := temp(j) ;
                    end if ;
               end loop ;
               bound := ( bound / 2 ) + ( bound mod 2 ) ;
          end loop ;
          output(d) <= temp(0)(7+NUM_INTERP_BITS
                         downto NUM_INTERP_BITS) ;
     end loop ;
end process ;
-------------------------------------------------
end nonsym_radial_tetrtl ;
-----------------------------------------
```

FIG.47C