NORGE

(12) **PATENT**

(19) NO (11) **314679** (13) B1

(51) Int Cl[7] H 04 N 7/30, G 06 K 9/36, G 01 V 1/28

Patentstyret

(21) Søknadsnr	19995448	(86) Int. inng. dag og søknadsnummer	1998.05.06, PCT/US98/09290
(22) Inng. dag	1999.11.05	(85) Videreføringsdag	1999.11.05
(24) Løpedag	1998.05.06	(30) Prioritet	1997.05.07, US, 45915
(41) Alm. tilgj.	1999.12.28		
(45) Meddelt dato	2003.04.28		

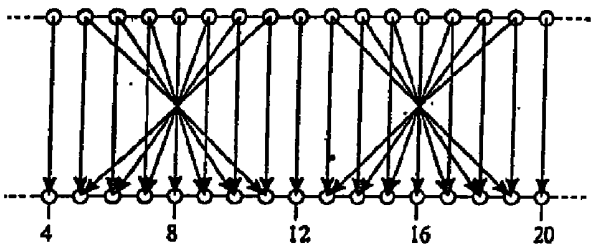
(71) Patenthaver Landmark Graphics Corp, 15150 Memorial Drive, Houston, TX 77079-4304, US
(72) Oppfinner Ira D Hale, Englewood, CO 80112, US
(74) Fullmektig Dag Thrane - ABC-Patent, Siving Rolf Chr B Larsen AS, 0602 Oslo

(54) Benevnelse **Fremgangsmåte for datakompresjon**

(56) Anførte publikasjoner Ingen

(57) Sammendrag

Den utbredte JPEG-standardalgoritme for to-dimensjonal bildekompresjon kan tilpasses kompresjon av grupper av enhver dimensjon og datatype, spesielt for grupper med seismiske data. Fordi JPEG-algoritmen behandler, mer eller mindre uavhengig, små delsett (8 x 8 blokker) av større bilder eller datagrupper, er slike tilpasninger spesielt nyttige ved anvendelser som ikke kan opprettholde en stor, ukomprimert, flerdimensjonal gruppe i datalageret. JPEG-lignende fremgangsmåter muliggjør kompresjon og dekompresjon av store grupper ved iterasjon over delgrupper som er små nok til å kunne befinne seg i lageret. Disse algoritmene fører til konseptet med et komprimert virtuelt lager. Spesielle hensyn må tas i den JPEG- lignende algoritme for å unngå blokk- artifakter, som er diskontinuiteter mellom blokker med data som er komprimert og dekomprimert uavhengig. Beregningsmessig effektive fremgangsmåter for undertrykkelse av disse artifaktene er heldigvis kjent. Av disse fremgangsmåtene har man anvendt den som gjør den mulig å gjenbruke meget av JPEG-fremgangsmåten. Den JPEG- lignende fremgangsmåte ifølge oppfinnelsen benytter JPEG-fremgangsmåtene til diskret cosinus-transformasjon (selv om fremover- transformasjonen og den inverse transformasjon er reversert), og for Huffman-koding av de kvantifiserte transformasjonskoeffisienter. Fremgangsmåten skiller seg fra JPEG hovedsakelig ved ytterligere trinn som foretas for å unngå blokkartifakter, og ved kvantifiseringen av trans- formasjonskoeffisientene.



Oppfinnelsen vedrører generelt en fremgangsmåte for komprimering av data, særlig for komprimering av en endimensjonal gruppe med sampler som er representative for karakteristikker i undergrunnen, og spesielt en forbedring av JPEG-metoden for bildekompresjon slik den blir anvendt på seismiske data.

JPEG-standardalgoritmen for bildekompresjon (Pennebaker og Mitchell, 1993) består av følgende tre trinn, utført for hver 8 x 8 blokk med piksler (bildepunkt eller bildeelement) i en todimensjonal gruppe:

1. Transformere 8 x 8 blokken med piksler med bruk av en diskret cosinus-transformasjon.
2. Kvantisere (skalere og avrunde) transformasjonskoeffisienten til små heltall.
3. Kode bitene ved å bruke få bits til å representere de hyppigste heltall.

Dekompresjonsalgoritmen inverterer hvert av disse trinn i motsatt rekkefølge. Begge algoritmer kan lett utvides til å komprimere og dekomprimere grupper av enhver dimensjon.

Fig. 1 viser en to-dimensjonal gruppe med seismiske data som ikke er blitt komprimert. Den to-dimensjonale gruppen på 32 bits med tall med flytende komma på fig. 1 er et tidskonstant utsnitt trukket ut fra en tre-dimensjonal seismisk undersøkelse. Fig. 2 viser et forstørret delsett av den samme gruppe.

Fig. 3 viser det samme forstørrede delsett etter kompresjon og dekompresjon av hele den to-dimensjonale gruppe ved bruk av en JPEG-lignende algoritme. Kompresjonsforholdet for hele gruppen er omkring 103:1, noe som betyr at den opprinnelige gruppe med 32 bits tall med flytende komma inneholdt omkring 103 ganger så mange bits som den komprimerte gruppe.

For slike store kompresjonsforhold frembringer denne JPEG-lignende algoritmen de blokkartifakter som er synlige på fig. 3. Ved lavere kompresjonsforhold blir disse diskontinuiteter mellom blokker mindre synlige, men de kan fremdeles

være betydelige, spesielt når ytterligere behandling eller tolkning blir utført etter kompresjon.

Artifaktene på fig. 3 er resultatet av at hver blokk på 8 x 8 sampler blir komprimert og dekomprimert uavhengig, uten noe forsett på å opprettholde kontinuiteten mellom blokkene.

Til tross for disse artifaktene er evnen til å komprimere og dekomprimere slike små delsett med data uavhengig, en ønskelig egenskap. Spesielt muliggjør det tilgang til en liten del av en stor komprimert gruppe uten å dekomprimere hele gruppen. Det gjør også kompresjonsalgoritmen i stand til å tilpasse seg romlige variasjoner i dataamplitude og spektrum. Disse egenskapene mangler i kompresjonsmetoder basert på småbølge-transformasjoner (f.eks. Bradley m.fl., 1993; Wickerhauser, 1994). Det problem som tas opp i denne beskrivelsen, er å oppnå disse egenskapene uten blokkartifaktene.

En løsning på dette problemet er å komprimere data ved bruk av overlappende blokker, slik at dekomprimerte sampelverdier kan beregnes uten referanse til verdier for tilstøtende blokkgrenser. Denne løsningen ble brukt av Yeo og Liu (1995) i deres tilpasning av JPEG-algoritmen til volumgjengivelse av tre-dimensjonale medisinske bilder. Bruken av overlappende blokker øker dessverre antallet blokker som må komprimeres, noe som øker beregningstider og minsker kompresjonsforholdene.

Henvisninger til tidligere arbeider sitert i denne beskrivelsen.

Bradley, J.N., C.M., and Hopper, T., 1993, The FBI wavelet/scalar quantization standard for gray-scale fingerprint image compression: Visual Information Processing II, SPIE Proceedings, pp. 293-304. (ftp://ftp.c3.lanl.gov/pub/WSQ).

Jawerth, B., and Sweldens, W., 1995, Biorthogonal smooth local trigonometric bases: J. Fourier Anal. Appl., 2; http://cm.bell-labs.com/who/wim/papers/papers.html.

Jawerth, B., Liu, Y., and Sweldens, W., 1996, Signal compression with smooth local trigonometric bases: http://cm.bell-labs.com/who/wim/papers/papers.html.

Malvar, H.S., and Staelin, D.H., 1989, The LOT-transform coding without blocking effects: IEEE Transactions of Acoustic, Speech and Signal Processing, 37, no. 4, 553-559.

Malwar, H.S., 1990, Lapped transform for efficient transform/subband coding; IEEE Transactions on Acoustic, Speech and Signal Processing, 38, no. 6, 969-978.

Pennebaker, W.B., and Mitchell, J.L., 1993, JPEG still image data compression standard: Van Nostrand Reinhold.

Princen, J.P., and Bradley, A.B., 1956, Analysis/synthesis filter bank design based on time domain aliasing cancellation: IEEE Transaction on Acoustics, Speech and Signal Processing, 34, no. 5, 1153-1161.

Wickerhauser, M.V., 1994, Adapted wavelet analysis from theory to software: A.K. Peters.

Yeo, B., and Liu, B., 1995, Volume rendering of DCT-based compressed 3D scalar data: IEEE Transactions on Visualization and Computer Graphics, 1, no. 1, 29-43.

Det er et hovedformål med oppfinnelsen å tilveiebringe en forbedret JPEG-algoritme for datakompresjon.

Et viktig formål med oppfinnelsen er å tilveiebringe en forbedret JPEG-datakompresjonsalgoritme som kan brukes til å komprimere små delsett med data uten blokkartifakter.

Oppfinnelsen angår en fremgangsmåte for datakompresjon som benytter JPEG-fremgangsmåter til diskret cosinustransformasjon og til Huffman-koding av de kvantiserte transformasjonskoeffisienter. Fremgangsmåten er en forbedring

i forhold til standard JPEG-fremgangsmåter, hovedsakelig i tilleggstrinnene for å unngå blokkartifakter, og i kvantiseringen av transformasjonskoeffisienter.

Dette oppnås ved å benytte en fremgangsmåte i henhold til de nedenfor fremsatte patentkrav.

Det vises til de vedføyde tegninger, hvor:

Fig. 1 representerer et tidskonstant utsnitt trukket ut fra en tre-dimensjonal seismisk undersøkelse, hvor fremvisningen er en to-dimensjonal gruppe med 32 bits tall med flytende komma som ikke er blitt komprimert;

fig. 2 er et forstørret delsett av den to-dimensjonale gruppe på fig. 1;

fig. 3 viser data på fig. 2 etter kompresjon og dekompresjon via en enkel tilpasning av JPEG-algoritmen, som viser blokkartifaktene, diskontinuitetene mellom 8 x 8 sampelblokkene som ble komprimert uavhengig;

fig. 4 viser data på fig. 2 etter kompresjon og dekompresjon via en JPEG-lignende algoritme som er blitt utformet for å undertrykke blokkartifakter;

fig. 5 viser en matrise C_{III}^{-1} som svarer til en invers DCT-III med lengde 32 hvor sorte piksler svarer til positive verdier; hvite piksler svarer til negative verdier;

fig. 6 viser en matrise som svarer til en blokkformet DCT-III med blokklengde 8;

fig. 7 viser kolonner 11 og 19 fra den inverse blokkformede DCT-III matrisen på fig. 6, hvor blokkartifaktene i JPEG-kompresjonen er forårsaket av diskontinuiteter, slik som dem som er vist mellom samplene 15 og 16, og det første sampel i hver cosinus ligger i avstand fra den plottede kurve og i denne figuren på grunn av diskontinuiteten i funksjonen b(j) definert ved ligning (1c) nedenfor;

fig. 8 viser glatt konusformede cosinuser oppnådd ved utfolding av de avkortede cosinuser som er vist på fig. 7, som er kolonnene 11 og 19 i den inverse foldede DCT-III-matrisen på fig. 9;

fig. 9 viser en matrise svarende til en invers foldet DCT-III hvor kolonnene i denne matrisen overlapper hverandre fra blokk til blokk, med verdier som glatt skrår til null slik at en veid sum av disse kolonnene ikke vil oppviser blokkartifakter;

fig. 10 viser en matrise svarende til den operasjon som brukes for å utfolde den inverse blokkformede DCT-III, hvor matrisen på fig. 9 er lik produktet av den matrisen og den som er vist på fig. 6;

fig. 11 viser en utfoldingsoperasjon som er en parvis kombinasjon og erstatning av sampelverdier over DCT-blokkgrenser, og hvor folding er det inverse av denne operasjonen; og

fig. 12 illustrerer trinnene for å dekomprimere det utpekte sampel i blokk A, som krever at man først dekoder, dekvantiserer og inverterer DCT-III-blokkene A, B, C og D, og så folder ut de fire utpekte samplene hvor foldings- og utfoldingsoperasjonene er en parvis blanding av sampler over DCT-blokkgrensene.

Oppfinnelsen angår en forbedret fremgangsmåte til data-kompresjon basert på sammenblanding av data i tilstøtende blokker under kompresjon med en JPEG-lignende algoritme. JPEG-lignende datakompresjon er blitt beskrevet i forskjellige former av mange forfattere (f.eks. Princen og Bradley, 1986; Malvar og Staelin, 1989; Malvar, 1990; Wickerhauser, 1994, Jawerth m.fl., 1995). Fordelen med JPEG-lignende datakompresjon er at den tenderer til å øke kompresjonsforhold, og bare svakt øker beregningstidene.

Fig. 4 viser det samme delsett fra den to-dimensjonale gruppe på fig. 1, etter kompresjon og dekompresjon ved bruk av en annen JPEG-lignende algoritme basert på fremgangsmåten ifølge oppfinnelsen som er beskrevet nedenfor. Kompresjons-

forholdet for hele gruppen er omkring 112:1. Blokkartifakter i disse dekomprimerte data er fraværende.

Forskjellen mellom den tidligere JPEG-lignende algoritmen og den nye JPEG-lignende algoritmen i henhold til oppfinnelsen ligger i en modifikasjon av den diskrete blokk-cosinus-transformasjon som er spesifisert i JPEG-standarden. Denne modifikasjonen i algoritmen ifølge oppfinnelsen demper blokkartifakter, mens de ønskede egenskaper ved den tidligere kjente JPEG-lignende algoritmen beholdes.

Den transformasjon som er brukt ved tidligere kjente JPEG-kompresjon er en diskret cosinus-transformasjon, kalt DCT-II. I henhold til oppfinnelsen blir det brukt en annen diskret cosinus-transformasjon kalt DCT-III. (For nummerering av diskrete cosinus-transformasjoner, se Wickerhauser, 1994, side 84).

Fremover DCT-III er definert ved

$$z(k)=\sum_{j=0}^{M-1} y(j)b(j)\sqrt{\frac{2}{M}}\cos\left[\frac{\pi(2k+1)j}{2M}\right] \qquad (1a)$$
$$k=0,1,\ldots,M-1$$

og den tilsvarende inverse transformasjon er

$$y(j)=\sum_{k=0}^{M-1} z(k)b(j)\sqrt{\frac{2}{M}}\cos\left[\frac{\pi(2k+1)j}{2M}\right] \qquad (1b)$$
$$j=0,1,\ldots,M-1,$$

hvor

$$b(j)\equiv\begin{cases}\frac{1}{\sqrt{2}}, & j=0\\ 1, & \textit{ellers}\end{cases} \qquad (1c)$$

Fremover-transformasjonen og den inverse transformasjon kan representeres som matrisemultiplikasjoner, som i $z = C_{III}y$ og $y=C_{III}^{-1}z$, hvor matrisen C_{III} har elementer

$$C_{III}(k,j) = b(j)\sqrt{\frac{2}{M}}\cos\left[\frac{\pi(2k+1)j}{2M}\right] \qquad (2)$$

Den inverse transformasjonsmatrise C_{III}^{-1} er illustrert på fig. 5, M=32. Enhver vektor med 32 reelle tall kan representeres som en veid sum av kolonnene i denne matrisen.

I en en-dimensjonal kompresjonsalgoritme kan en vektor y med sampelverdier tilnærmes med bare noen få kolonner fra matrisen C_{III}^{-1}. Vektene for hver kolonne vil være gitt av transformasjonskoeffisienter i vektoren z. Høye kompresjonsforhold krever at mange av koeffisientene i z kan overses, nær null. Slike små koeffisienter vil bli kvantifisert til nuller som effektivt kan kodes ved bruk av få bits.

JPEG-standarden beskriver kompresjon av to-dimensjonale bilder, ikke en-dimensjonale vektorer. Den tidligere DCT-II som brukes i JPEG-kompresjon er en to-dimensjonal transformasjon. Fordi den diskrete coninus-transformasjon av flerdimensjonale data kan utføres som en kaskade av en-dimensjonale transformasjoner langs hver datadimensjon, blir imidlertid bare den en-dimensjonale transformasjon beskrevet for enkelhets skyld.

For lange datavektorer, slik som seismiske traser, er det lite sannsynlig at én enkelt diskret cosinus-transformasjon av hele vektoren vil gi mange transformasjonskoeffisienter som er neglisjerbare. I likhet med JPEG blir det derfor brukt en blokkoppdelt DCT med transformasjonslengde M=8, og fylldatavektorer blir satt til null, etter behov, til en lengde N som er en multippel av 8. Matrisen som svarer til en invers blokk-DCT-III er vist på fig. 6, for N=32 og M=8. Foruten å være mer passende for kompresjon, er blokktransformasjonen også mer effektiv med en beregningskostnad som bare vokser lineært med lengden N av datavektorene.

I likhet med tidligere kjente DCT-II som brukes i JPEG-kompresjon, har DCT-III flere nyttige egenskaper. For det første transformerer en DCT-III reelle tall til reelle tall, slik at ingen kompleks aritmetikk er nødvendig. I likhet med

den tidligere kjente DCT-II, er også DCT-III en enhetstransformasjon: $C_{III}^{-1} = C_{III}^{T}$. [Denne egenskapen er gitt ved ligningene (1) ovenfor]. Fremover blokk-DCT-III-matrisen som svarer til fig. 6, er ganske enkelt den transponerte av den inverse blokk-DCT-III som er vist her.

En annen nyttig egenskap er at en invers DCT-III er ekvivalent med en fremover DCT-II: $C_{III}^{-1} = C_{II}$. Dette forholdet mellom DCT-III og DCT-II og et valg av en transformasjonslengde M=8, muliggjør bruk av de høyt optimaliserte DCT-II-algoritmer som brukes i JPEG-kompresjon, ved enkel ombytting av fremover algoritmene og de inverse algoritmer.

Alle de ovenfor nevnte egenskaper er nyttige, men hvorfor ikke ganske enkelt bruke den tidligere kjente DCT-II i JPEG? Svaret ligger i en modifikasjon i henhold til oppfinnelsen av DCT-III for å unngå blokkartifaktene som forårsakes av JPEG-kompresjon.

Blokkartifakter opptrer når bare et delsett av kolonnene i matrisen som er vist på fig. 6, blir brukt i en tilnærmelse til en vektor. Anta f.eks. at en slik vektor er sterkt komprimert ved bruk av bare to kolonner med indekser k=11 og k=19. Som vist på fig. 7 gir enhver ikke-triviell kombinasjon av disse to kolonnene en diskontinuitet mellom sampelindeksene j=15 og j=16 i tilnærmelsen. Slike diskontinuiteter skaper blokkartifaktene som er synlige i bilder som er blitt komprimert ved bruk av JPEG-algoritmen.

For å unngå de artifakter som er forårsaket av bruk av en blokk-DCT-III ved kompresjon, blir blokk-cosinusene som er illustrert på fig. 7, erstattet med de glatt avskrådde cosinuser som er vist på fig. 8. Sammenlign den fullstendige inverse transformasjonsmatrise som er vist på fig. 9 med den inverse blokk-DCT-III-matrise som er vist på fig. 6. Antallet sampler i hver cosinus har øket til 16 (bortsett fra ved endene) slik at cosinuser i tilstøtende blokker nå overlapper hverandre, og at hver cosinus glatt avskrås til null. En veid sum av disse cosinusene vil ikke frembringe den diskontinuitet som er vist på fig. 7.

Transformasjonene som svarer til disse avskrådde cosinuser, blir ofte kalt lokale cosinus-transformasjoner eller overlappede ortogonale transformasjoner (Wickerhauser, 1994, side 370; det er en likhet mellom fig. 8 og Wickerhausers fig. 11.5). Her blir disse transformasjonene kalt foldede cosinus-transformasjoner for å avspeile den måte de beregnes på. Den transformasjon som brukes i kompresjonsalgoritmen i henhold til oppfinnelsen er spesielt en foldet DCT-III. Den inverse transformasjon som svarer til den matrise som er vist på fig. 9, kalles en invers foldet DCT-III.

Folding er en måte til å oppnå glatte, avskrådde, 16-samplers cosinuser ved bruk av meget optimaliserte, blokkformede, M=8 DCT-algoritmer. Wickerhauser (1994, side 103) beskriver denne fremgangsmåten som "en bemerkelsesverdig observasjon gjort uavhengig av flere individer", og fortsetter med å diskutere dens anvendelse i forbindelse med kompresjon. Foldingsoperasjonen som brukes ved kompresjon i denne oppfinnelse er én av mange som er beskrevet av Wickerhauser, men ble inspirert av arbeidene til Jawerth og Sweldens (1995) og Jawerth m.fl. (1996). De sistnevnte forfattere diskuterer aspekter ved folding som er spesielt relevante for kompresjon.

Smartheten og effektiviteten ved folding ligger i det faktum at den inverse foldede DCT-III-matrise som er vist på fig. 9, er produktet av en utfoldingsmatrise, vist på fig. 10, og den inverse blokk-DCT-III-matrise som er vist på fig. 6. Fordi hver rad og kolonne i utfoldingsmatrisen ikke inneholder mer enn to elementer forskjellig fra null, er beregningskostnadene ved utfolding et nesten ubetydelig tillegg til kostnadene ved den inverse blokk-DCT-III.

I praksis er de matriser som er vist på fig. 6, 9 eller 10 aldri konstruert i virkeligheten. I stedet blir operasjoner utført på sampelverdier som har den samme virkning som multiplikasjon ved hjelp av disse matrisene. For den inverse blokk-DCT-III-matrisen på fig. 6, er denne operasjonen den meget effektive fremover blokk-DCT-II-algoritmen som brukes i

JPEG-kompresjon. For utfoldingsmatrisen på fig. 10 er denne operasjonen en parvis blanding av sampelverdier over blokkgrensene ved indeksene j=8, 16,..., som vist på fig. 11.

Foldingsoperasjonen er ganske enkelt det inverse av utfoldingsoperasjonen, en annen parvis blanding av de samme sampelverdier. Fordi folding og utfolding er sentrert omkring blokkgrenser, er disse operasjonene mest konsist spesifisert uttrykt ved forskjøvede sampelverdier, som er definert ved $y_l(j) \equiv y(|M+j)$. Symbolet l er en blokkindeks og j er et sampel inne i en blokkindeks.

Folding blir så utført via

$$\begin{aligned} y_l(j) &= f(j)\, x_l(j) + f(-j)\, x_l(-j), \\ y_l(-j) &= f(j)\, x_l(-j) - f(-j)\, x_l(j), \\ l &= 1,2,\ldots,\ N \mid M - 1, \\ j &= 1,2,\ldots,\ M \mid 2 - 1, \\ y_l(j) &= x_l(j), \quad \textit{ellers}, \end{aligned} \tag{3a}$$

og utfolding blir utført via

$$\begin{aligned} x_l(j) &= f(j)\, y_l(j) - f(-j)\, y_l(-j), \\ x_l(-j) &= f(j)\, y_l(-j) + f(-j)\, y_l(j), \\ l &= 1,2,\ldots, N \mid M - 1, \\ j &= 1,2,\ldots, M \mid 2 - 1, \\ x_l(j) &= y_l(j), \quad \textit{ellers}, \end{aligned} \tag{3b}$$

hvor foldingsfunksjonen f(j) er definert ved

$$f(j) \equiv \sin\left[\frac{\pi}{4}\right], \tag{3c}$$

De glatt avskrådde cosinuser på fig. 8 og matrisen til disse cosinusene på fig. 9 ble beregnet ved anvendelse av ligningene (3b) på kolonnene i den matrise som er vist på fig. 6.

Den fremover foldede DCT-III av samplede verdier x(j) blir bestemt ved først å beregne y(j) via ligningene (3a), og

så for hver blokk på åtte sampler, beregning av z(k) via ligning (1a). Likeledes blir den inverse foldede DCT-III bestemt ved først å beregne y(j) via ligning (1b) for hver blokk, og så beregne x(j) via ligningene (3b).

Som diskutert av Wickerhauser (1994, side 105), Jawerth og Sweldens (1995) og Jawerth m.fl. (1996), er mange alternative, men lignende foldingsoperasjoner mulige. Den foldingsoperasjonen som er definert ved ligningene (3), er valgt av to grunner.

For det første er foldingsoperasjonen enhetlig. Utfoldingsmatrisen som er vist på fig. 10 er den transponerte av den tilsvarende foldingsmatrise (ikke vist). For å verifisere denne egenskapen, kan det legges merke til at foldingsfunksjonen i ligning (3c) tilfredsstiller $f^2(j) + f^2(-j) = 1$, og uttrykker så foldings- og utfoldingsoperasjonene i ligningene (3a) og (3b) som multiplikasjoner av 2 x 2 matriser. Inverter analytisk 2 x 2 foldingsmatrisen for å se at dens inverse er lik dens transponerte, som er lik 2 x 2 utfoldingsmatrisen. Hele foldingsoperasjonen er derfor enhetlig, fordi den består av disse 2 x 2 blandingene av sampelverdier over blokkgrenser.

For det annet sikrer foldingsoperasjonen i henhold til oppfinnelsen at en konstant funksjon, slik som x(j)=1, gir transformasjonskoeffisienter $z_1(k)$ i hver blokk som er forskjellig fra null for bare k=0, noe som forsterker kompresjonen av konstante (eller svakt varierende) data. Ifølge terminologien til Jawerth m.fl. (1996), oppnår den fremover foldede DCT-III "oppløsning av konstantene".

For å verifisere denne annen egenskap ved den foldede DCT-III, anvend analytisk foldingsoperasjonene i ligningene (3a) på konstante sampelverdier $x_1(j)=1$, og verifiser at resultatet er $y_1(j)= \sqrt{M} C_{III}(0j)$, hvor $C_{III}(k,j)$ er definert i ligning (2). Foldingsfunksjonen blir med andre ord valgt slik at konstante sampelverdier blir foldet for nøyaktig å passe til (innenfor en skalafaktor $\sqrt{M}$) den første (k=0) cosinus i DCT-III. Fordi denne cosinus er ortogonal til alle andre

cosinuser i transformasjonen, vil bare transformasjonskoeffisienten k=0 i hver blokk være forskjellig fra null etter den fremover foldede DCT-III. Verdien av hver koeffisient som er forskjellig fra null, vil være $z_1(0)=\sqrt{M}$.

Mer nøyaktig gjelder denne egenskapen for alle unntatt den første og den siste blokken. Selv om oppløsning av konstantene for de første og siste blokker kan oppnås ved å modifisere foldings- og utfoldingsmatrisene slik at de ikke er enhetlige i sine hjørner, blir mindre kompresjon akseptert i disse blokkene og en strengt enhetlig foldings- og utfoldingsoperasjon blir opprettholdt.

Den fremover DCT-II som brukes i JPEG-kompresjon, oppnår oppløsning av konstantene uten folding, fordi den første cosinus i DCT-II (f.eks. den første rad i matrisen på fig. 5) er konstant. Denne egenskapen er i virkeligheten grunne til at DCT-II, ikke DCT-III, er spesifisert i JPEG-kompresjonsstandarden. Bruk av foldingstrikket med DCT-II kan tenkes, fordi folding før JPEG-kompresjon og utfolding etter JPEG-dekompresjon kan utføres uten å modifisere JPEG-standardalgoritmen.

Foldings- og utfoldingsoperasjonene i ligningene (3) er dessverre ikke riktige for DCT-II. Spesielt ville ligningene (3b) ikke gi glatt avskrådde cosinuser lik de som er vist på fig. 8.

Jawerth m.fl. (1995) beskriver alternative foldings- og utfoldingsoperasjoner som er anvendbare for DCT-II, og som opprettholder oppløsning av konstantene. Disse operasjonene er imidlertid ikke enhetlige. De er i virkeligheten dårlig tilpasset og har en tendens til å forsterke diskontinuiteter i sampelverdier som inntreffer nær blokkgrenser, og reduserer dermed effektiviteten av kompresjonen.

Folding- og utfoldingsoperasjonene ifølge ligningene (3) er derimot enhetlige i likhet med den fremover DCT-III og den inverse DCT-III i følge ligningene (1). Hvis F betegner foldingsmatrisen, så kan den fremover foldede DCT-III uttrykkes som $z=C_{III}Fx$ og legg merke til at

$$
\begin{aligned}
z^T z &= x^T F^T C_{III}^T C_{III} F x \\
&= x^T F^{-1} C_{III}^{-1} C_{III} F x \\
&= x^T x \\
\sum_{k=0}^{N-1} z^2(k) &= \sum_{j=0}^{N-1} x^2(j)
\end{aligned}
\qquad (4)
$$

Summen av kvadrerte sampelverdier etter foldet DCT-III er med andre ord lik summen før den foldede DCT-III. Denne egenskapen kan brukes til å anslå forvrengningen i de dekomprimerte data forårsaket ved kvantisering av transformasjonskoeffisientene.

Som beskrevet ovenfor er den transformasjon som brukes i kompresjonsalgoritmen ifølge oppfinnelsen nettopp det inverse av den som brukes i JPEG-kompresjon, men med folding innbefattet for å redusere blokkartifakter. Kvantifiserings- og kodingsmetoder som benyttes, er også tilpasset fra de som brukes ved JPEG-kompresjon. Forskjellene mellom JPEG-fremgangsmåtene og oppfinnelsen er beskrevet nedenfor.

Fordi JPEG-kompresjon er ment for bilder, er data før kompresjon enten 8 bits eller 12 bits verdier. (Fargebilder er representert ved rød-grønn-blå-tripletter av slike sampler). Etter en to-dimensjonal transformasjon via DCT-II, kan 8 bits data kreve opptil 11 bits pr. sampel fordi den største sampelverdi etter to-dimensjonal DCT-II er opptil 8 ganger større enn før transformasjonen. Generelt er den største sampelverdi etter DCT-II opp til $M^{D/2}$ ganger den største verdi før transformasjonen, hvor D er antallet transformerte dimensjoner. Med andre ord

$$
|z|_{max} \leq M^{D/2} |x|_{max} \qquad (5)
$$

Denne øvre grense blir oppnådd når data nøyaktig tilsvarer én av de cosinuser som brukes i DCT-II, slik som den første konstante cosinus $C_{II}(k=0,j)$.

Den samme faktor $M^{D/2}$ gjelder også den foldede DCT-III som brukes i kompresjonsalgoritmen ifølge oppfinnelsen. Etter transformering av 8 bits heltallsdata med en foldet DCT-III, kvantiserer og koder derfor fremgangsmåten ifølge oppfinnelsen 11 bits heltall i to-dimensjonal kompresjon, og 13 bits heltall i tre-dimensjonal kompresjon.

Kvantiseringstrinnet i både JPEG-kompresjonsalgoritmen og fremgangsmåten ifølge oppfinnelsen er en skaleringsoperasjon utformet for å redusere antallet bits som skal kodes, og det er denne reduksjonen av antallet bits som er ansvarlig for den forvrengning i data som er komprimert og så dekomprimert. I slike tapsbringende kompresjonsmetoder blir denne forvrengningen akseptert i bytte med høye kompresjonsforhold.

Ved JPEG-kompresjon blir transformasjonskoeffisientene i hver blokk kvantisert forskjellig, med høye bølgetall (romfrekvenser) representert med færre bits enn lave bølgetall. Denne bølgetall-avhengige skalering i JPEG-kompresjon blir vanligvis optimalisert med hensyn på menneskelig visuell oppfattelse.

Ved kompresjonsmetoden ifølge oppfinnelsen blir bølgetall alle kvantisert likt. Feil blir ikke innført som er bølgetall-avhengige. En grunn til dette er at for seismiske data, er det ofte stor interesse for høye bølgetall. F.eks. svarer undergrunnsforkastninger avbildet med seismiske data til høye bølgetall. En annen grunn er at seismiske data ofte blir analysert ved hjelp av datamaskinalgoritmer som er uavhengige av det menneskelige visuelle system.

Ved komprimering av 8 bits bildedata ligger sampelverdier mellom -128 og +127; og det kan antas at lavamplitudeblokker med data er ubetydelige og trygt kan kvantiseres til null under kompresjon. JPEG-standarden gjør spesielt denne antagelsen, for den tillater bare ett sett med kvantiseringsskala-faktorer for et helt bilde. Som diskutert ovenfor kan disse skalafaktorene variere for forskjellige koeffisienter (forskjellige bølgetall) innenfor en blokk, men det samme

sett blir brukt for hver blokk. I denne betydning er JPEG-kvantisering global.

Ved komprimering av 32 bits data med flytende komma blir det foretrukket å utføre en lokal kvantisering, med skala-faktorer som kan variere fra blokk til blokk. Før komprimering av slike data er eventuelt den maksimale sampelamplitude $|x|_{max}$ ikke kjent, og avlesning av hvert sampel før kompresjon for å bestemme dens verdi, kan være kostbar. Lave amplituder kan videre ikke med rette antas å være ubetydelige; spesielt seismiske data krever ofte betydelig behandling før denne antagelsen er gyldig. Selv om en enkelt skalafaktor blir brukt til å kvantisere alle transformasjonskoeffisienter innenfor en blokk, kan derfor en skalafaktor tillates å variere fra blokk til blokk. Spesielt i ligning (5) ovenfor betegner $|z|_{max}$ maksimumskoeffisienten innenfor hver transformert blokk, og en forskjellig skalafaktor s blir beregnet for hver blokk.

Lokalt kvantisering gir lavere kompresjonsforhold (frembringer flere bits pr. sampel) enn global kvantisering. En opplagt grunn er at ytterligere bits er nødvendig for å lagre kvantiseringsskala-faktorene for hver komprimert blokk. En mindre opplagt grunn er at lokal kvantifisering kan kvantifisere færre sampler til null enn global kvantifisering. Et valg mellom enten lokal eller global kvantifisering er derfor tillatt i kompresjonsalgoritmen ifølge oppfinnelsen.

Den lokale kvantifisering håndterer ikke et stort dynamisk område innenfor en enkelt blokk. F.eks. kan refleksjoner med lav amplitude i ubehandlede seismiske data være gjemt under overflatebølger med høy amplitude. Innenfor blokker som er forurenset med støy med høy amplitude, kan kompresjons-algoritmen ifølge oppfinnelsen, selv med lokal kvantifisering, kvantifisere signaler med lav amplitude til null, slik at signalet ikke kan gjenfinnes ved behandling etter dekomprimering. Støy med høy amplitude bør derfor dempes før kompresjon.

Et virtuelt lager representerer illusjonen om et lager som er større enn det som er fysisk tilgjengelig. Denne illusjonen er mest effektiv for anvendelser som har tendens til å aksessere data anbragt nær andre data som nylig ble aksessert. Slike anvendelser har god referanseposisjon.

Anvendelser som arbeider med to-dimensjonale eller tre-dimensjonale grupper oppviser ofte god referanseposisjon. En seismisk tolkningsanvendelse kan f.eks. fremvise påfølgende to-dimensjonale utsnitt av data fra en tre-dimensjonal gruppe. Med en tilstrekkelig hurtig, lokal dekompresjons-algoritme, kan blokker dekomprimeres som inneholder samplene for slike utsnitt etter behov, uten å dekomprimere hele den tre-dimensjonale gruppe. Kompresjonsalgoritmen ifølge oppfinnelsen er spesielt nyttig i slike anvendelser.

Nøkkelen til slike anvendelser er evnen til å komprimere eller dekomprimere et delsett i en større gruppe uten å komprimere eller dekomprimere hele gruppen. For kompresjon basert på en blokk-DCT-II, lik den som brukes i JPEG-kompresjon, kommer denne egenskapen lett. For å dekomprimere et enkelt sampel er det spesielt bare nødvendig å dekode, dekvantifisere og invertere den DCT-II-blokken som inneholder vedkommende sampel. Straks dekompresjon for et sampel er utført, blir dekompresjon av alle samplene i dens blokk utført. Hvis det antas referanseposisjon, så kan beregningskostnadene ved dekomprimering av de andre samplene i blokken ikke være bortkastet.

For kompresjonsalgoritmen ifølge oppfinnelsen, basert på en foldet DCT-III, er det nødvendig med noe ytterligere arbeid. Betrakt to-dimensjonal kompresjon og de fire blokkene med 8 x 8 sampler som er illustrert på fig. 12. For å dekomprimere det sampel som svarer til den fylte sirkel i blokk A, er det nødvendig (1) å dekode, dekvantifisere og invertere alle fire DCT-III-blokkene (A,B,C og D), og (2) å utfolde de fire samplene som svarer til de fylte sirklene. Utfolding over blokkgrenser er akkurat som illustrert på fig. 11 og som beskrevet ved ligning (3b), utført først for én dimensjon og

så for den annen. Selv om fire blokker er nødvendig, er straks dekomprimeringen av sampelet i blokk A er utført, mesteparten av det arbeid som er nødvendig for å dekomprimere nabosampler, blitt utført. Hvis det igjen antas referanse-posisjon vil denne ytterligere beregning ikke være bort-kastet.

En annen forskjell mellom kvantifiseringstrinnet ifølge oppfinnelsen og JPEG stammer fra vårt behov for å kvantifisere 32 bits data med flytende komma, som har et meget høyere dynamisk område enn 8 bits eller 12 bits bildedata. For å kvantifisere en verdi z med flytende komma til et heltall i med B+1 bits (innbefattende en fortegnsbit), blir følgende algoritme brukt

$$i = \begin{cases} [z \times s + 1/2], & z \geq 0 \\ [z \times s - 1/2], & z > 0 \end{cases} \qquad (6)$$

hvor s er kvantifiseringsskala-faktoren.

For å unngå overflyt er $|i| < 2^B$ nødvendig. Denne restriksjonen og ligning (6) fører til følgende ligning for skalafaktoren:

$$s = \frac{(2^B - 1/2)(1 - \varepsilon)}{|z|_{max}} \qquad (7)$$

hvor ε er flyt-epsilon, det minste positive tall som kan subtraheres fra 1 ved bruk av aritmetikk med flytende komma, for å oppnå et tall forskjellig fra 1.

JPEG-kompresjonsstandarden tillater to fremgangsmåter til koding av heltallene som er frembragt ved kvantifisering, Huffman-koding og aritmetisk koding, og mange andre kodings-metoder er mulige. Huffman-koding i JPEG blir brukt i kompre-sjonsalgoritmen ifølge oppfinnelsen, fordi den er beregnings-messig hurtig, er enkel å implementere og er fritt tilgjenge-lig.

Med et unntak følger Huffman-kodings- og dekodings-algoritmer JPEG-spesifikasjonen. JPEG-standarden spesifiserer spesiell koding av DC-transformasjonskoeffisienter (k=0). JPEG-kompresjon utnytter det faktum at disse DC-koeffisient-ene ofte er høyt korrelert blant naboblokker. Denne spesielle behandling blir ikke brukt ifølge fremgangsmåten i henhold til oppfinnelsen av to grunner: (1) den innfører uavhengighet fra blokk til blokk som kompliserer kompresjon og dekompre-sjon av enkeltblokker, og (2) seismiske data har typisk forholdsvis små DC-koeffisienter. (Selv om de er små, er DC-koeffisientene sjelden neglisjerbare på grunn av den korte M=8 transformasjon som brukes). DC-koeffisienten blir derfor kodet akkurat som de andre (AC) koeffisientene blir kodet.

Hovedfordelen ved den JPEG-lignende algoritme er at endel av en fler-dimensjonal gruppe kan komprimeres eller brukes uten behandling av hele gruppen. Kompresjonsalgoritmer basert på småbølge-transformasjoner (f.eks. Bradley m.fl., 1933; Wickerhauser, 1994) mangler derimot denne egenskapen. Mens JPEG-standarden ikke eksplisitt understøtter denne egenskapen, gjør den blokk-DCT-II som brukes i JPEG-kompre-sjon det mulig. Denne evnen blir utnyttet i kompresjonsalgo-ritmen ifølge oppfinnelsen basert på en foldet DCT-III.

Hver datablokk i en gruppe kan tenkes på som analog med en side i et virtuelt lager. For to-dimensjonal kompresjon vil hver side inneholde 64 = 8 x 8 sampler; for tre-dimen-sjonal kompresjon ville hver side inneholde 512 = 8 x 8 x 8 sampler. Sampler blir dekomprimert etter hvert som de blir bladd inn, og bølgenummer komprimert etter hvert som de blir bladd ut etter bølgenummer. Et arbeidssett med ukomprimerte sider blir beholdt i lageret, mens mesteparten av sidene forblir komprimert og lagret enten i lageret eller på en plate. (Hvis de lagres på plate, kan sidene kombineres for å forbedre I/O-effektiviteten). Hvis arbeidssettet er stort nok, og hvis foreliggende oppfinnelse har god posisjons-referanse, vil beregningskostnadene ved komprimering og/eller

dekomprimering av hver side bli amortisert over tilgangen til flesteparten av prøvene innenfor disse sidene.

Tilpasningen av JPEG-kompresjonsalgoritmen i henhold til oppfinnelsen, gjør det mulig å gjenbruke meget av algoritmen. JPEG-metodene blir gjenbrukt for lange 8 DCTer, ved ganske enkelt å bytte om fremover-transformasjonene og de inverse transformasjonene. JPEG-kvantifiseringsmetoden er modifisert for å unngå referansebehandling av lave bølgetall, og for å håndtere variasjoner fra blokk til blokk i dataamplituder. Sampelverdier blir også foldet over blokkgrenser før en fremover DCT, og slike verdier blir utfoldet etter en invers DCT. Denne foldingen og utfoldingen undertrykker blokk-artifaktene som er synlige i bilder som er blitt komprimert med JPEG-algoritmen.

For å sammenligne ytelsen til fremgangsmåten ifølge oppfinnelsen med andre algoritmer, er to nyttige mål på ytelse beregningstid og forvrengning for et spesifisert kompresjonsforhold. Foreløpige fikspunkter med en tidlig implementering av algoritmen ifølge oppfinnelsen er oppmuntrende. Beregningstidene er omkring halvparten og forvrengningene er nesten identiske med de som gjelder for en småbølgebasert algoritme over et bredt område med kompresjonsforhold (Diller 1997, personlig kommunikasjon). Beregningstidene antas å være lavere for fremgangsmåten ifølge oppfinnelsen enn for småbølgebaserte fremgangsmåter (på grunn av færre multiplikasjoner og addisjoner og mer lokal lagerbruk).

P a t e n t k r a v

1. Fremgangsmåte for komprimering av en en-dimensjonal gruppe x med N sampler som er representative for karakteristikker i undergrunnen,

k a r a k t e r i s e r t v e d:

a) å inndele gruppen x i blokker med M sampler, hvor M<N,

b) å folde samplene $x_l(j) \equiv x(/M+j)$ over hver blokkgrense *l*, i henhold til

$$y_l(-j) = f(j)x_l(j) + f(-j)x_l(-j),$$
$$y_l(-j) = f(j)x_l(-j) - f(-j)x_l(j),$$
$$l = 1,2,\ldots,N/M-1,$$
$$j = 1,2,\ldots,M/2-1,$$
$$y_l(j) = x_l(j), \text{ ellers,}$$

hvor

$$f(j) \equiv \sin\left[\frac{\pi}{4}\left(1+\frac{2j}{M}\right)\right]$$

c) å transformere de foldede sampler i hver blokk i gruppen y i henhold til

$$z(k) = \sum_{j=0}^{M-1} y(j)b(j)\sqrt{\frac{2}{M}}\cos\left[\frac{\pi(2k+1)j}{2M}\right]$$

$$k = 0,1,\ldots,M-1,$$

hvor

$$b(j) \equiv \begin{cases} \frac{1}{\sqrt{2}}, & j=0 \\ 1, & \text{ellers} \end{cases}$$

d) å kvantifisere de transformerte sampler i hver blokk i gruppen z for å frembringe heltall, og

e) å kode heltallene til en strøm av bits som representerer den komprimerte gruppe.

2. Fremgangsmåte ifølge krav 1, anvendt på en flerdimensjonal gruppe,
k a r a k t e r i s e r t v e d at a) blokkoppdelings-, b) foldings- og c) transformeringstrinnene blir anvendt som en kaskade av operasjoner langs hver gruppedimensjon.

3. Fremgangsmåte ifølge krav 2,
k a r a k t e r i s e r t v e d at et delsett av den flerdimensjonale gruppe blir komprimert.

4. Fremgangsmåte ifølge krav 3,
k a r a k t e r i s e r t v e d at den flerdimensjonale gruppe representerer seismiske signaler.

5. Fremgangsmåte ifølge krav 1,
k a r a k t e r i s e r t v e d å dekomprimere den komprimerte gruppe ved å invertere trinnene a) til e) i motsatt rekkefølge.

6. Fremgangsmåte ifølge krav 5, anvendt på en flerdimensjonal gruppe,
k a r a k t e r i s e r t v e d at det inverse av a) blokkoppdelings-, b) foldings- og c) transformeringstrinnene blir anvendt som en kaskade av operasjoner langs hver gruppedimensjon.

7. Fremgangsmåte ifølge krav 6,
k a r a k t e r i s e r t v e d at et delsett av den flerdimensjonale gruppe blir dekomprimert.

8. Fremgangsmåte ifølge krav 7,
k a r a k t e r i s e r t v e d at den flerdimensjonale gruppe representerer seismiske signaler.

FIG. 1

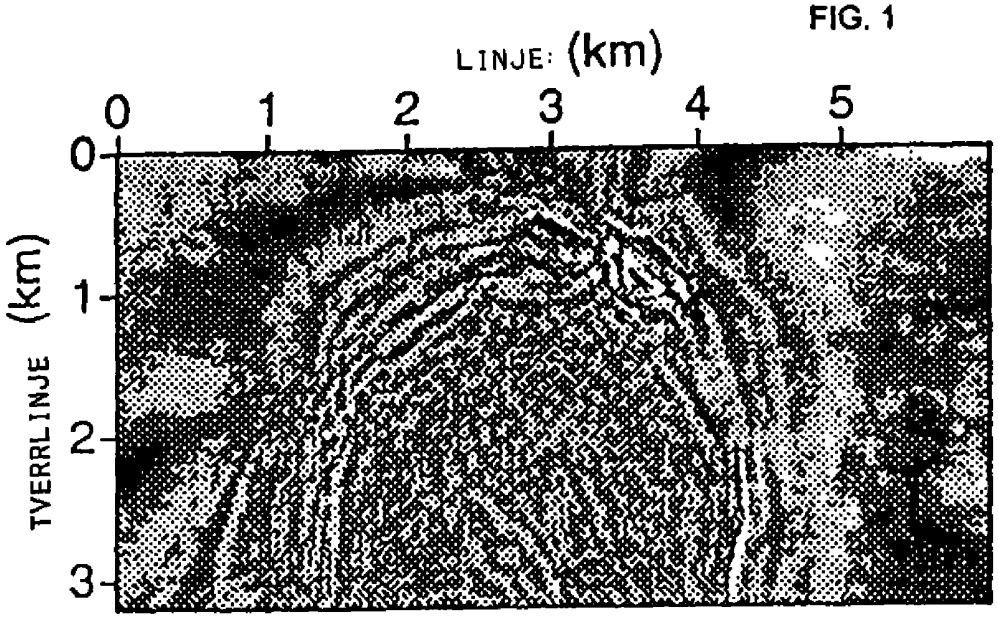
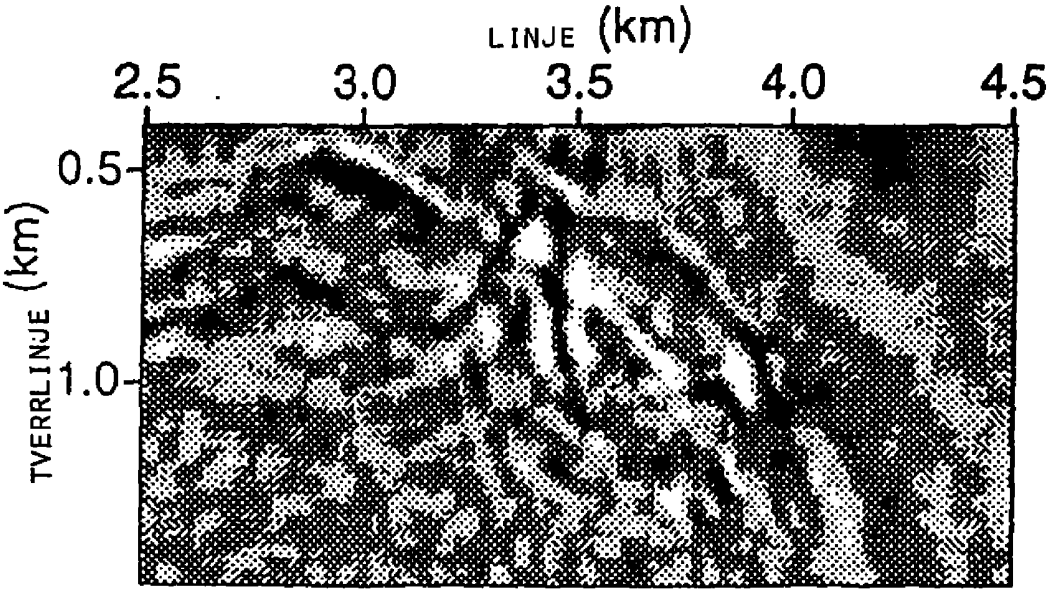

LINJE (km)
0
1
2
3
4
5
TVERRLINJE (km)
0
1
2
3

FIG. 2


LINJE (km)
2.5
3.0
3.5
4.0
4.5
TVERRLINJE (km)
0.5
1.0

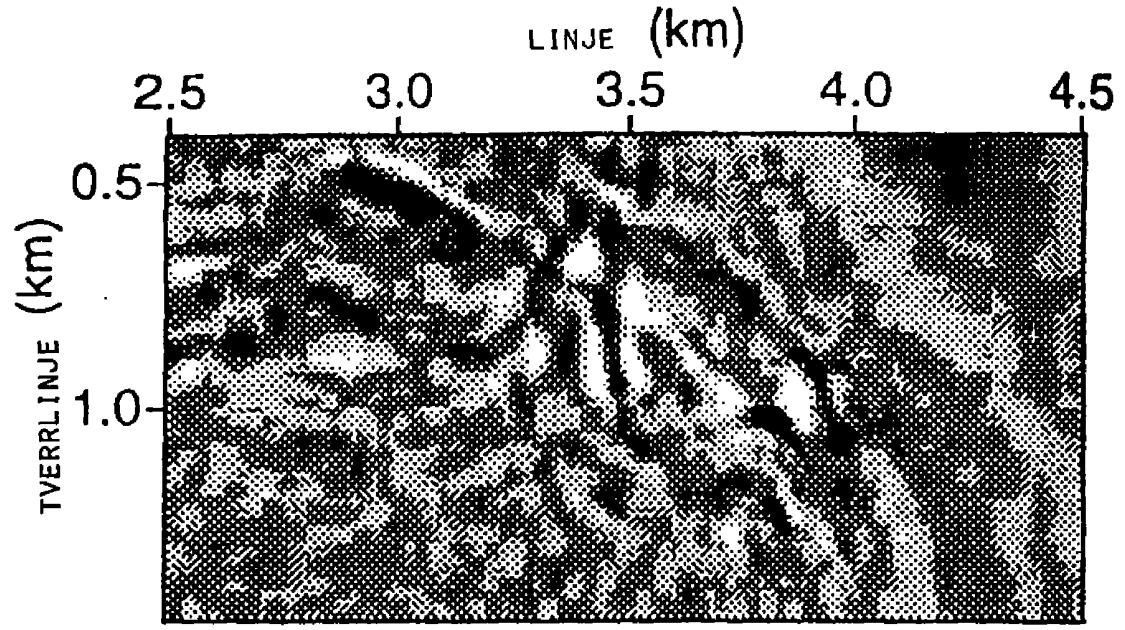

LINJE (km)
2.5
3.0
3.5
4.0
4.5
0.5
1.0
TVERRLINJE (km)

FIG. 3.

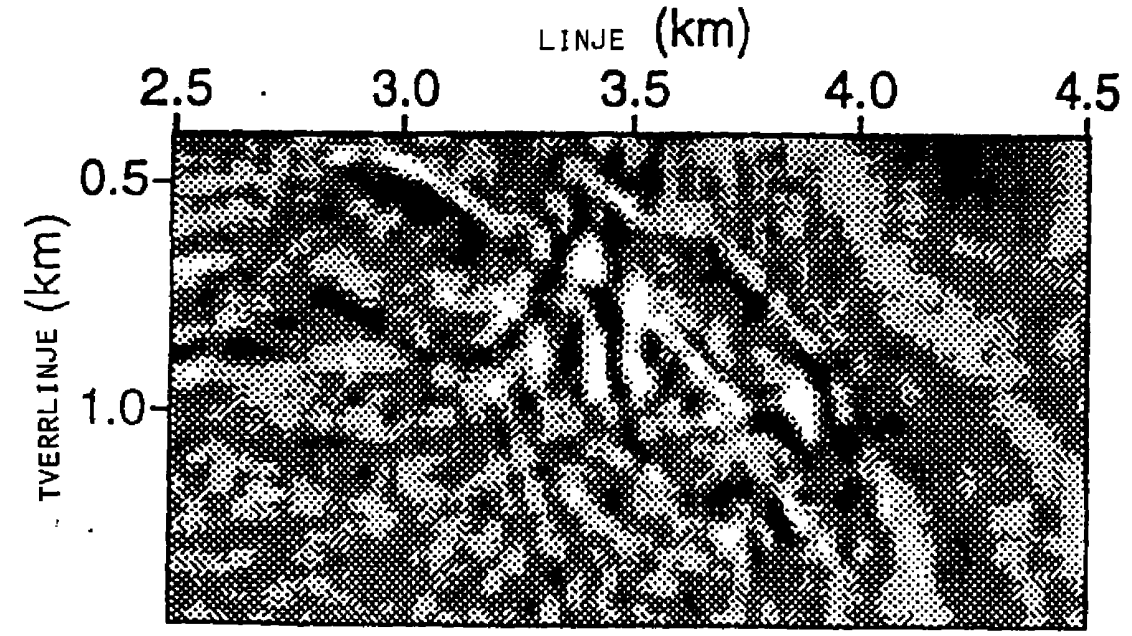

LINJE (km)
2.5
3.0
3.5
4.0
4.5
0.5
1.0
TVERRLINJE (km)

FIG. 4.

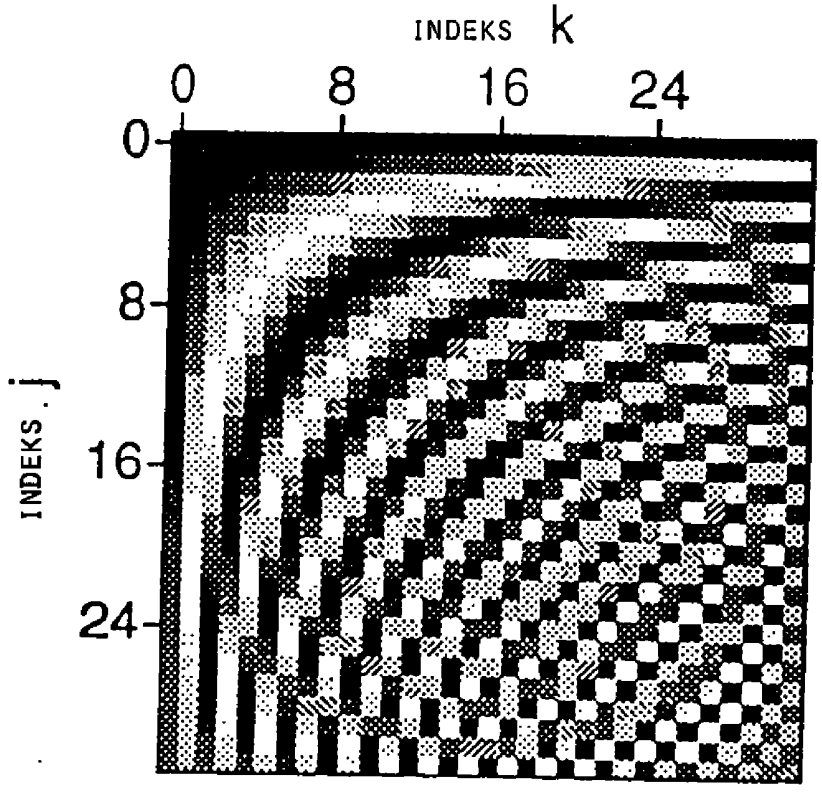

INDEKS k
0
8
16
24
INDEKS j
0
8
16
24

FIG. 5.

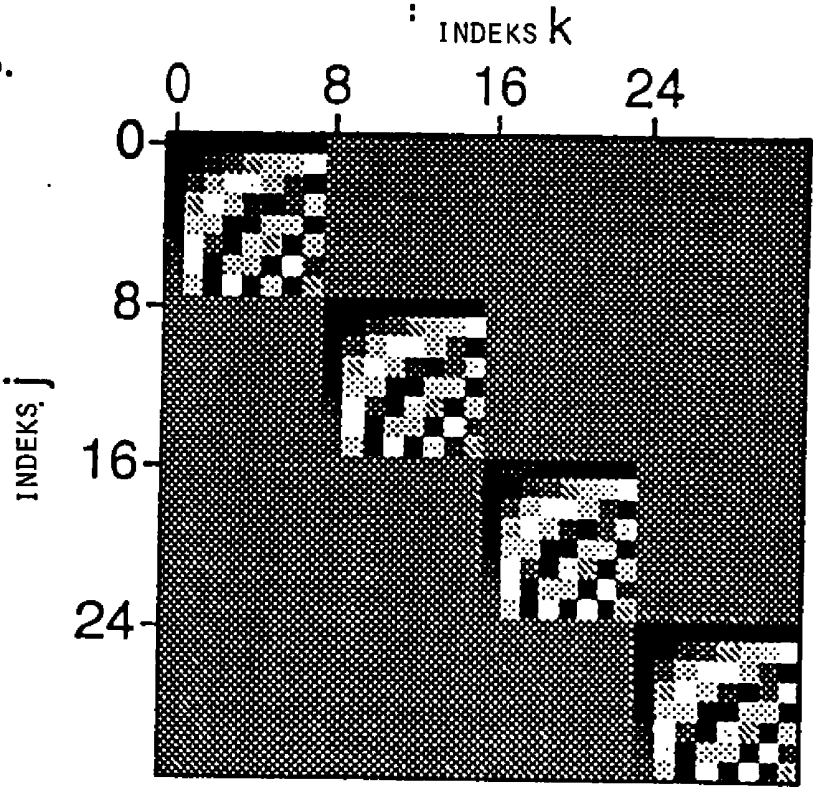

INDEKS k
0
8
16
24
INDEKS j
0
8
16
24

FIG. 6.

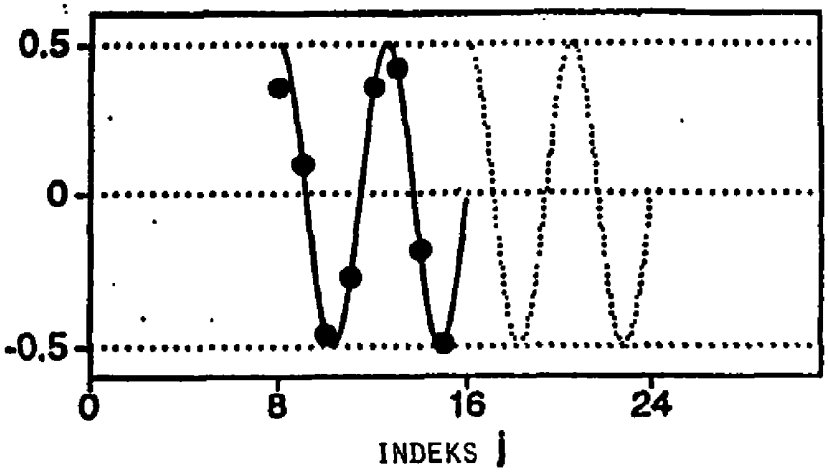


FIG. 7.

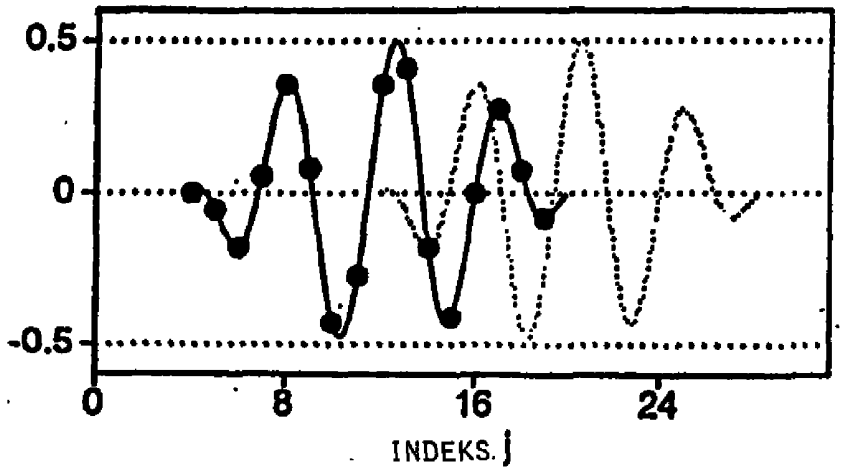


FIG. 8.

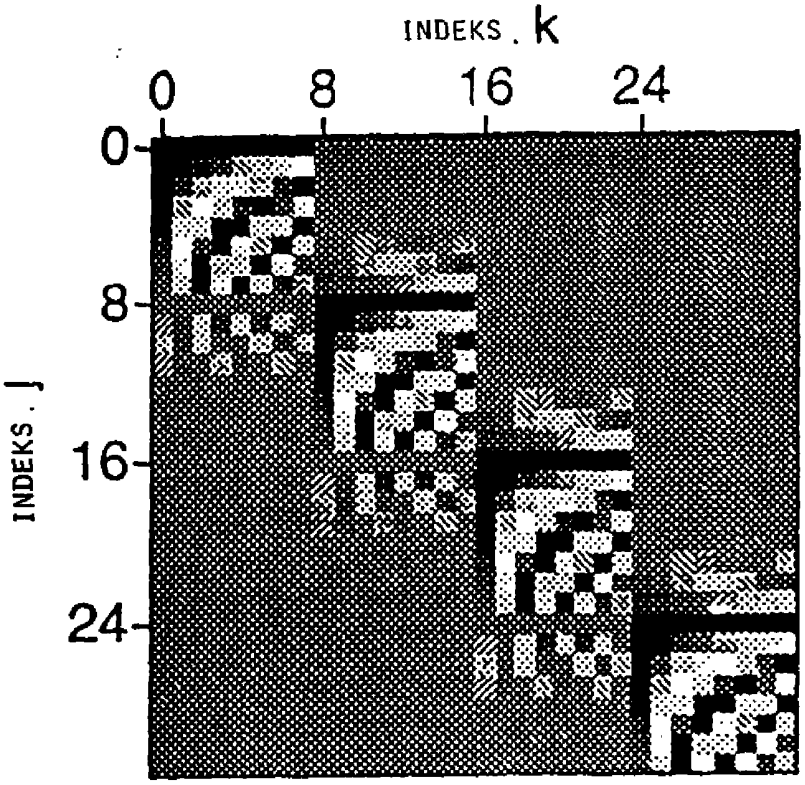

INDEKS . k
0
8
16
24
INDEKS . j
0
8
16
24

FIG. 9.

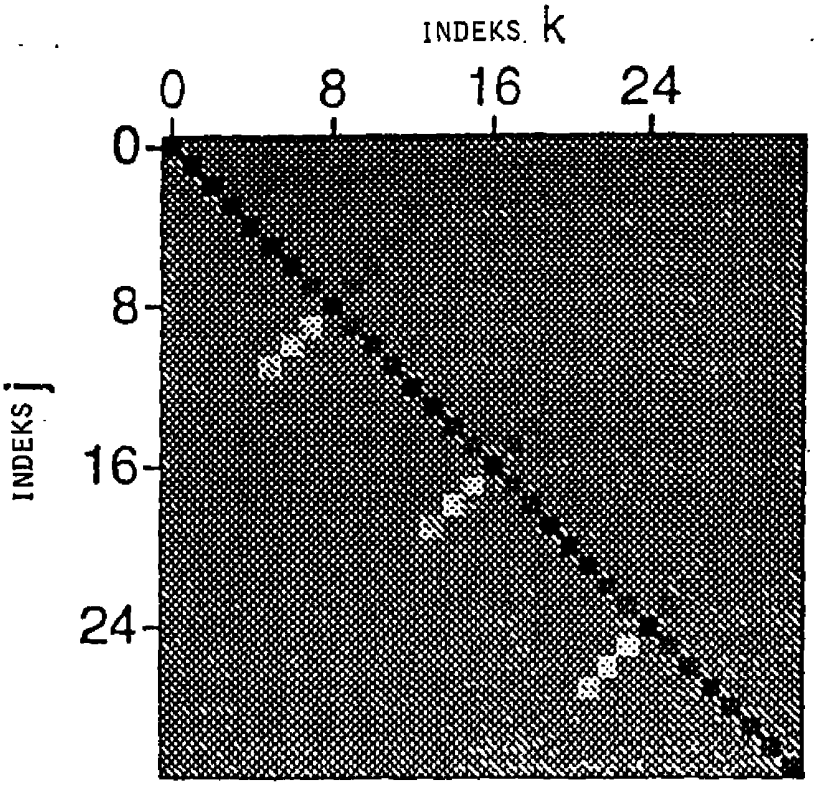

INDEKS. k
0
8
16
24
INDEKS j
0
8
16
24

FIG. 10.

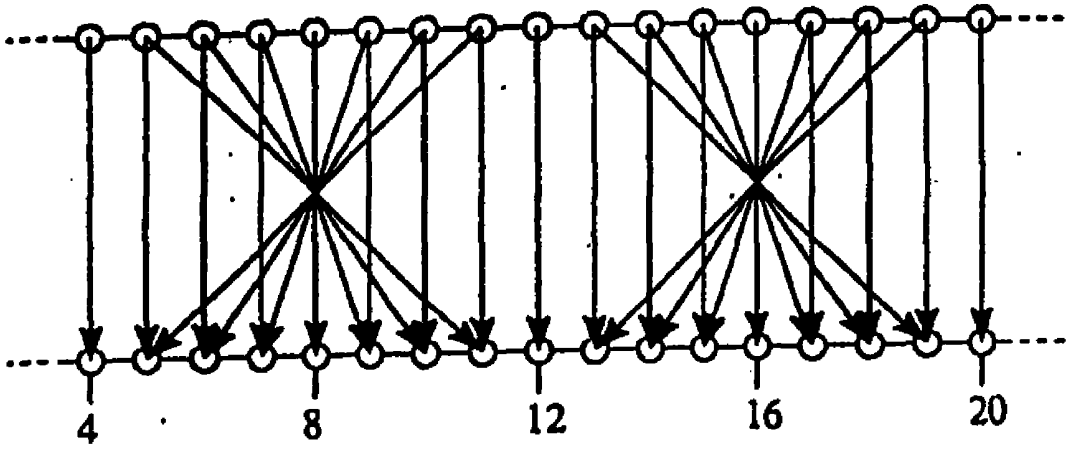


FIG. 11.

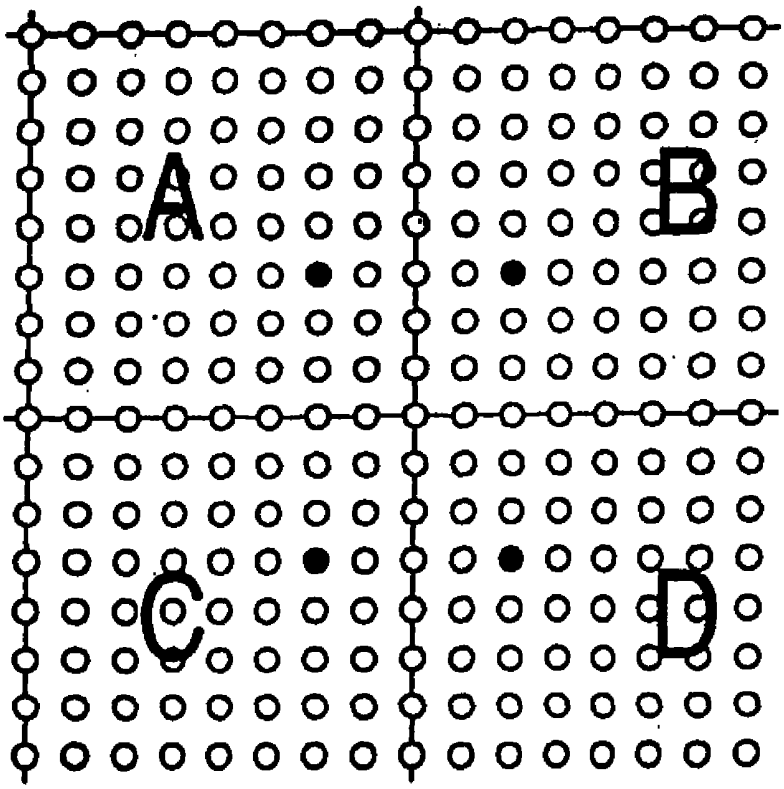


FIG. 12.