KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

Saulius Styga

PROGRESYVIOJO VAIZDŲ KODAVIMO SPEKTRINĖJE HAAR TRANSFORMACIJOS SRITYJE METODO REALIZACIJA IR TYRIMAS

Magistro darbas

Vadovas
prof. dr. J. Valantinas

KAUNAS, 2012

KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. N. Listopadskis
2012 06 02

PROGRESYVIOJO VAIZDŲ KODAVIMO SPEKTRINĖJE HAAR TRANSFORMACIJOS SRITYJE METODO REALIZACIJA IR TYRIMAS

Taikomosios matematikos magistro baigiamasis darbas

Vadovas
prof. dr. J. Valantinas
2012 06 01

Recenzentas
doc. dr. R.Marcinkevičius
2012 06 01

Atliko
FMM 0 gr. stud.
Saulius Styga
2012 05 30

KAUNAS, 2012

KVALIFIKACINĖ KOMISIJA

Pirmininkas: Rimantas Rudzkis, habil. dr., vyriausiasis analitikas (DnB NORD Bankas)

Sekretorius: Eimutis Valakevičius, docentas (KTU)

Nariai: Jonas Valantinas, profesorius (KTU)

Vytautas Janilionis, docentas (KTU)

Vidmantas Povilas Pekarskas, profesorius (KTU)

Zenonas Navickas, profesorius (KTU)

Arūnas Barauskas, dr., vice-prezidentas projektams (UAB „Baltic Amadeus“)

Styga S. Method realization and analysis of progressive image coding in spectral domain of Haar transform.: Master‘s work in applied mathematics / supervisor prof. dr. J. Valantinas; Department of Applied mathematics, Faculty of Fundamental Sciences, Kaunas University of Technology. – Kaunas, 2012. – 92 p.

SUMMARY

Image is the key tool to present graphical information. The more graphical information it contains, the easier it is to absorb this information, but a significant amount of graphical information will consume IT resources and slows down the transmission speed without the need. In order to reduce the waste of resources in information graphics and speed up the information transmission speed, it is proposed to use image coding technologies. One of these technologies is a locally progressive image coding. Its essence is to improve only the selected fragment quality of the image. Also, this technology is an integral part of the discrete wavelet transform.

On this paper new original procedure for the evaluation of discrete Haar spectra for separate fragments of a digital image is proposed and examined. The procedure explores specific properties of Haar wavelets, refers to the assumption that Haar spectrum of the whole image is known, and this procedure results is compared with direct evaluation of Haar spectral coefficients for the respective image blocks procedure results. Also Haar wavelet features and discrete Haar transform and its fast computing algorithm discovered.

At this paper we also examine one of the most effective image compression algorithms SPIHT. This algorithm works in discrete wavelet transform spectral domain but we are facing one major issue with Haar spectrum: this spectrum elements are real numbers but SPIHT works only with integer numbers. In order to implement SPIHT with Haar spectra, Haar transform has been modified to produce integer numbers instead of real.

Some experiment results are also presented. They corroborate out theoretical conclusions that SPIHT is partly able to work with Haar transform.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

ĮVADAS

Vaizdas yra pagrindinė priemonė pateikti grafinei informacijai. Kuo daugiau jame grafinės informacijos, tuo lengviau šią informaciją įsisavinti, tačiau per didelis kiekis grafinės informacijos be reikalo eikvoja informacinių technologijų resursus bei sulėtina informacijos perdavimo greitį. Siekiant sumažinti grafinės informacijos eikvojamus resursus bei paspartinti informacijos perdavimo greitį, siūloma naudoti vaizdo kodavimo technologijas. Viena šių technologijų yra lokaliai progresyvus vaizdų kodavimas. Jos esmė yra dekodavimo metu gerinti tik pasirinkto vaizdo fragmento kokybę. Taip pat, ši technologija yra neatsiejama nuo diskrečiųjų bangelių transformacijų.

Šiame darbe nagrinėsime labai greitą perėjimo nuo Haar spektro visam vaizdui prie Haar spektro pasirinktam vaizdo fragmentui procedūrą ir gautus rezultatus bei skaičiavimo greitį palyginsime su tiesioginės vaizdo fragmento Haar transformacijos gautais rezultatais ir skaičiavimo greičiu. Taip pat, susipažinsime su Haar bangelių savybėmis ir diskrečiąja Haar transformacija bei greitu jos skaičiavimo algoritmu.

Pasinaudoję SPIHT algoritmu, nagrinėsime vaizdo suspaudimo efektyvumą diskrečiosios Haar transformacijos srityje. Spręsime Haar transformacijos nesuderinamumo su SPIHT algoritmu problemą, kadangi SPIHT dirba tik su sveikaisiais skaičiais, o Haar diskrečioji transformacija yra sudaryta iš realiųjų skaičių. Tam modifikuosime vieną iš greitų Haar transformacijos algoritmų – Andrews.

Palyginsime Haar transformacijos, rastos naudojant modifikuotą Andrews algoritmą, apskaičiavimo laiką su Haar transformacijos, rastos naudojant nemodifikuotą Andrews algoritmą, laiku.

Pagrindinis darbo tikslas – pratiškai suderinti teoriškai nesuderinamus Haar transformaciją ir SPIHT metodą bei kiek įmanoma efektyviau progresyviai užkoduoti vaizdą.

1. DISKREČIOSIOS BANGELIŲ TRANSFORMACIJOS (DBT)

Diskrecioji bangelių transformacija (DBT) gimė vystant diskrečiąją Furjė transformaciją (DFT), kai pastarojoje buvo pastebėti trūkumai, kurių pagrindinis – lokalizavimo erdveje problema. DBT atsiradimas leido šią problemą išspręsti. DBT metu, apdorojant skaitmeninį signalą, natūraliai suformuojamas kintamo dydžio (mastelio) langas, kuris slenka išilgai laiko (erdvės) ašies. Kiekvienoje lango pozicijoje atliekama transformacija ir tokiu būdu randamas bangelių spektras. Šis procesas kartojamas daug kartų, parenkant trumpesnius ar ilgesnius langus. Galutinis proceso rezultatas yra signalo išraiškų (vaizdavimų pagal dažnį ir laiką) rinkinys.

Kiekvienoje diskrečiosios bangelių transformacijos iteracijoje (DBT) įvesties duomenims (skaitmeniniam signalui) yra naudojama mastelio funkcija, kurios pagalba apskaičiuojama pusė suvidurkintų signalo X reikšmių. Čia N, kur $N = 2^n, n \in N$, yra signalo x reikšmių skaičius. Gautos reikšmės yra saugomos viršutinėje įvesties vektoriaus iš N elementų dalyje.

Kita pusė skaitmeninio signalo X reikšmių taip pat skaičiuojama kiekviename diskrečiosios bangelių transformacijos žingsnyje. Tam taikoma taip vadinama bangelės funkcija, kurią naudojant randamos signalo X skirtuminės (parodo informacijos pokyčius) reikšmės. Šios reikšmės yra saugomos apatinėje įvesties vektoriaus iš N elementų dalyje.

Sekančios iteracijos metu mastelio ir bangelės funkcijos naudojamos tik suvidurkintoms skaitmeninio signalo X rekšmėms, gautoms prieš tai atliktos iteracijos metu. Iteracija atliekama n kartų, tai yra tol, kol įvesties vektoriuje lieka vienintelė suvidurkinta reikšmė. Atlikus n žingsnių iš skaitmeninio signalo X gaunamas diskrečiųjų bangelių spektras Y. Nesunku pastebėti, kad spektras Y apima sutvarkytą skirtuminių reikšmių rinkinį ir vienintelę suvidurkintą reikšmę (pirmasis vektoriaus Y elementas).

1.1. BENDROSIOS BANGELIŲ SAVYBĖS

Pirmiausia apibrėšime tolydžiąją bangelių transformaciją (TBT), tai yra:

$$\gamma(s,\tau) = f(x)\Psi^*_{s,\tau}(x)dx \qquad \textbf{(1.1)}$$

čia žymėjimas * reiškia kompleksinę junginę išraišką. Ši lygtis parodo, kaip funkcija $f(x)$ yra išskaidoma į bazinių funkcijų (bangelių) $\Psi_{s,\tau}(x)$ aibę. Kintamasis s (mąstelis) ir

kintamasis τ (poslinkis) yra po bangelių transformacijos gautos koordinatės. Išsamumo dėlei, apibrėšime atvirkštinę bangelių transformaciją, tai yra:

$$f(x) = \iint \gamma(s,\tau)\Psi_{s,\tau}(x)d\tau ds \quad \textbf{(1.2)}$$

Bangelės yra generuojamos iš pagrindinės (motininės) bangelės $\Psi(x)$, naudojant mastelio keitimą bei poslinkį:

$$\Psi_{s,\tau}(x) = \frac{1}{\sqrt{s}}\Psi\left(\frac{x-\tau}{s}\right) \quad \textbf{(1.3)}$$

čia s yra mastelį keičiantis koeficientas, τ – poslinkio koeficientas, o $\frac{1}{\sqrt{s}}$ - energijos normalizavimo koeficientas, atsižvelgiant į skirtingas mastelio koeficiento reikšmes.

Svarbu paminėti, kad (1.1), (1.2) ir (1.3) formulėse bazinės bangelių funkcijos yra neapibrėžtos. Tai yra didžiausias skirtumas tarp bangelių transformacijos ir Furjė ar kitų transformacijų. Bangelių transformacijų teorijoje nagrinėjamos tik bendrosios bangelių ir jų transformacijų savybės. Sudaroma struktūra, kurią naudojant leidžiama pasirinkti bangeles pagal poreikį.

Pagrindinės bangelių savybės yra tinkamumo ir reguliarumo. Būtent šios dvi savybės įtakojo pavadinimo „bangelės“ atsiradimą. Kvadratu integruojama funkcija $\Psi(x)$ tenkina tinkamumo sąlygą, jeigu:

$$\int \frac{|\Psi(\omega)|^2}{|\omega|}d\omega < +\infty \quad \textbf{(1.4)}$$

ši funkcija gali būti panaudota pirma signalo analizavimui, o tada ir jo rekonstravimui, kurios metu duomenys neprarandami. Čia $\Psi(\omega)$ yra funkcijos $\Psi(x)$ Furjė transformaciją.

Reguliarumo sąlyga, kuri išplaukia iš (1.1.4) nelygybės, reiškia, kad Furjė transformacija $\Psi(x)$ išnyksta, kai dažnis ω yra lygus nuliui, tai yra:

$$|\Psi(\omega)|^2|_{\omega=0} = 0 \quad \textbf{(1.5)}$$

Ši savybė parodo, kad bangelės privalo turėti dažnių juostą kaip spektre. Taip pat, bangelės pastovioji dedamoji lygi nuliui, tai yra:

$$\int \Psi(x)dx = 0 \quad \textbf{(1.6)}$$

Kitais žodžiais, $\Psi(x)$ privalo būti bangelė.

Iš (1.1) formulės matyti, kad vienmatės funkcijos bangelių transformacija yra dvimatė, dvimatės funkcijos – keturmatė ir taip toliau.

Bangelėms (bangelių funkcijoms) dažnai iškeliamos papildomos sąlygos, kurios pagerina bangelių skleidinių konvergavimo greitį. Šios sąlygos susijusios su bangelių reguliarumo savybe. Ši savybė yra tarsi reikalavimas, kad bangelė būtų pakankamai glodi ir sukoncentruota laiko bei dažnių srityse. Reguliarumo savybė yra gana sudėtinga sąvoka. Tam, kad šią savoką paaiškintume ir tinkamai interpretuotume, pasinaudosime nykstančiais pradiniais momentais.

Jeigu, paprastumo dėlei, prilygintume $\tau = 0$ ir bangelių transformaciją (1.1) išskleistume Teiloro eilute taško $x = 0$ aplinkoje, gautume:

$$\gamma(s,0) = \frac{1}{\sqrt{s}}\left[\sum_{p=0}^{n} f^{(p)}(0) \int \frac{t^p}{p!} \Psi\left(\frac{x}{s}\right) dt + O(n+1)\right] \quad \textbf{(1.7)}$$

čia $f^{(p)}$ yra p – toji funkcijos f išvestinė, o $O(n+1)$ yra skleidinio liekamasis narys. Pažymėję bangelės momentus M_p šitaip:

$$M_p = \int t^p \Psi(x) dt \quad \textbf{(1.8)}$$

galime perrašyti (1.7) išraišką taip:

$$\gamma(s,0) = \frac{1}{\sqrt{s}}\left[f(0)M_0 s + \frac{f^{(1)}(0)}{1!} M_1 s^2 + \frac{f^{(2)}(0)}{2!} M_2 s^3 + \cdots + \frac{f^{(n)}(0)}{n!} M_n s^{n+1} + O(s^{n+2})\right] \quad \textbf{(1.9)}$$

Iš tinkamumo salygos turime, kad $M_0 = 0$, taigi, pirmasis (1.9) išraiškos narys taip pat bus lygus nuliui. Jeigu ir kitus momentus M_n, kur $n = 1, 2, \ldots, N; n \in N$; N- momentų skaičius, padarytume lygius nuliui, tuomet tolydžiam signalui $f(x)$ bangelių transformacijos koeficientai $\gamma(s,\tau)$ sloptų greičiu $s^n + 2$. Toki momentai yra vadinami nykstančiais momentais. Jų reikšmės ne būtinai turi būti lygios nuliui, dažniausiai užtenka, kad šios reikšmės būtų artimos nuliui.

Apibendrinus bangelių savybes, galima teigti, kad tinkamumo sąlygos įvykdymas įtakoja bangelės svyravimus, o reguliarumo sąlygos įvykdymas ir nykstantys momentai – greitesnį bangelių skleidinių konvergavimą.

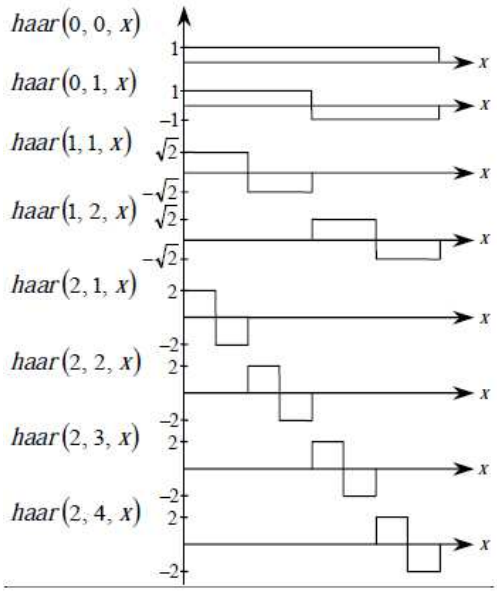
1.2. HAAR BANGELĖS IR JŲ SPECIFIKA

Haar bangelės pripažįstamos kaip pirmosios žinomos bangelės ir yra vienos iš paprasčiausių bangelių, lyginant su kitomis. Dažniausiai jos naudojamos kaip prototipas visoms kitoms bangelių rūšims. Haro bangelių ortogonali sistema apibrėžiama taip:

$$haar(0,0,x) = 1; \tag{1.10}$$

$$haar(r,m,x) = \begin{cases} 2^{\frac{r}{2}}, & \frac{m-1}{2^r} \le x < \frac{m-\frac{1}{2}}{2^r} \\ -2^{\frac{r}{2}}, & \frac{m-\frac{1}{2}}{2^r} \le x < \frac{m}{2^r} \\ 0, & kitu\ atveju \end{cases} \tag{1.11}$$

su visais visais $r = 0,1,\dots,log_2(N-1)$ ir $m = 1,2,\dots,2^r$; $x \in [0;1)$.



1.1 pav. Haar bangelės, kai N = 8

Be to, pastebėkime, kad Haar motininės bangelės funkciją $\psi(x)$ galima užrašyti šitaip:

$$\psi(x) = \begin{cases} 1, & 0 \le x < \frac{1}{2}, \\ -1, & \frac{1}{2} \le x < 1, \\ 0, & kitu\ atveju \end{cases} \quad \textbf{(1.12)}$$

Tuo tarpu, Haar mastelio funkciją $\varphi(x)$ turi tokį pavidalą:

$$\varphi(x) = \begin{cases} 1, & 0 \le x < 1, \\ 0, & kitu\ atveju \end{cases} \quad \textbf{(1.13)}$$

Motininės bangelės ir mastelio funkcijos turi tokias savybes:

1. Bet kuri funkcija gali būti aproksimuota tiesiniais funkcijų $\varphi(x)$, $\varphi(2x)$, ... , $\varphi(2^k x)$ ir $\psi(x)$, $\psi(2x)$, ... , $\psi(2^k x)$ dariniais.
2. Išpildoma ortogonalumo sąlygą, tai yra:

$$\int_{-\infty}^{\infty} 2^m \psi(2^m x - n)\psi(2^{m_1} x - n_1)dt = \delta_{m,m_1}\delta_{n,n_1} \quad \textbf{(1.14)}$$

čia $\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \ne j \end{cases}$. **(1.15)**

3. $\varphi(x) = \varphi(2x) + \varphi(2x-1)$ ir $\psi(x) = \varphi(2x) - \varphi(2x-1)$ **(1.16; 1.17)**

Aptarsime funkcijų $f(x)$ skleidinius ortogonalios Haar funkcijų sistemos (1.10 ir 1.11) atžvilgiu. Šiuose skleidiniuose informacija lokalizuojama ir pagal dažnį, ir erdvėje. Paimkime funkciją $f(x)$, $x \in [0; 1)$ ir užrašykime ją Furjė eilute Haar funkcijų sistemos $\{haar(r, m, x)\}$ atžvilgiu. Pirmiausia, įveskime tokį pažymėjimą:

$$i = 2^r + m \quad \textbf{(1.18)}$$

Imdami vieną indeksą vietoje dviejų, gausime sistemą $\{haar(i, x)\}$, kur $i = 1, 2, \ldots$; $x \in [0; 1)$. Tada Haar skleidinys funkcijai $f(x)$ bus toks:

$$f(x) = d_1 haar(1, x) + d_2 haar(2, x) + d_3 haar(3, x) + \ \ldots = \sum_{i=1}^{\infty} d_i haar(i, x) \quad \textbf{(1.19)}$$

čia d_i žymi skleidinio koeficientus, kurie randami pagal formulę:

$$d_i = \int_0^1 f(x) haar(i,x)dx \text{ , } i = 1,2,\dots \text{ .} \quad \textbf{(1.20)}$$

Haro skleidinį (1.1.10) galima užrašyti kitokia forma. Tam įvesime tokius pažymėjimus: $\varphi(x) = haar(0,0,x)$ ir $\psi(x) = haar(0,1,x)$, kur $x \in [0;1)$. Tuomet Haar skleidinys bus toks:

$$f(x) = a_0 + a_{00}\psi(x) + a_{10}\psi(2x) + a_{11}\psi(2x-1) + a_{20}\psi(4x) + a_{21}\psi(4x-1) + a_{22}\psi(4x-2) + a_{23}\psi(4x-3) + a_{30}\psi(8x) + \dots = a_0 + \sum_{r=0}^{\infty}\sum_{m=0}^{2^r-1} a_{rm}\psi(2^r x - m) \quad \textbf{(1.21)}$$

su visais $r = 0,1,2,\dots$ ir $m = 0,1,\dots,2^r - 1$. Čia a_0 ir a_{rm} yra Haar koeficientai, kur $a_0 = d_1$, o $a_{rm} = d_{2^r+k+1}$.

Matome, kad visos Haar funkcijos sistemoje $\{haar(i,x)\}$, kur $i = 1,2,\dots$; $x \in [0;1)$, pradedant antrąja, gali būti generuojamos motininei Haar bangelei $\psi(x) = har(0,1,x)$, $x \in [0;1)$, pritaikius diadinius suspaudimus bei sveikaskaitinius poslinkius išilgai x ašies.[1]

Haar eilutės dalinė suma

$$S_n''(x) = a_0 + \sum_{r=0}^{n}\sum_{m=0}^{2^r-1} a_{rm}\,\psi(2^r x - m) \quad \textbf{(1.22)}$$

yra funkcijos (signalo) $f(x)$ aproksimacija, kuri įvertina ne žemesnės kaip 2^{-n} eilės detales.

Haar eilutės puikiai lokalizuotos erdvėje. Tarkim, kad reikia ištirti atkarpą $[a,b] \subset [0;1)$. Tam užtenka paimti ir nagrinėti tik tuos Haar skleidinio (1.21) koeficientus a_{rm}, kurių indeksai r ir m išpildo sąlygą

$$I_{rm} = [2^{-r}m; 2^{-r}(m+1)] \cap [a,b] \neq \varnothing \quad \textbf{(1.23)}$$

tai yra, su kuriais Haar bangelė $\psi(2^r x - m)$ tapačiai nelygi nuliui atkarpoje $[a,b]$.

Lokalizavimo erdvėje savybė labai supaprastina spektrinę funkcijų (fizikinių signalų, realaus pasaulio vaizdų ir panašiai) analizę, daro ją efektyvesne.[1]

1.3. DISKREČIOJI HAAR TRANSFORMACIJA (HT) IR JOS RADIMAS

1.3.1. VIENMATIS ATVEJIS

Tarkime, turime vektorių $X = [X(m)] = (X(0)X(1) \dots X(N-1))^T$, kur $N = 2^n$, $n \in N$. Diskrečioji Haar transformacija (HT) šiam vektoriui apibrėžiama tokiu būdu:

$$Y_X = (Y(0)Y(1) \dots Y(N-1))^T = \frac{1}{N} H(n) \cdot X \qquad \textbf{(1.24)}$$

čia vektorius Y_X yra vektoriaus X HT spektras, o $H(n)$ – HT matrica, kurios matmenys $N \times N$. Ji gaunama diskretizuojant Haar funkcijų aibę $\{haar(r, m, x)\}$, tai yra, funkcijas (1.10) ir (1.11). Pavyzdžiui, kai $N = 8$, tai HT matrica $H(3)$ įgaus tokį pavidalą:

$$H(3) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{pmatrix} \begin{matrix} \} \frac{N}{2} \\ \\ \\ \\ \} \frac{N}{4} \\ \\ \\ \\ \end{matrix} \qquad \textbf{(1.25)}$$

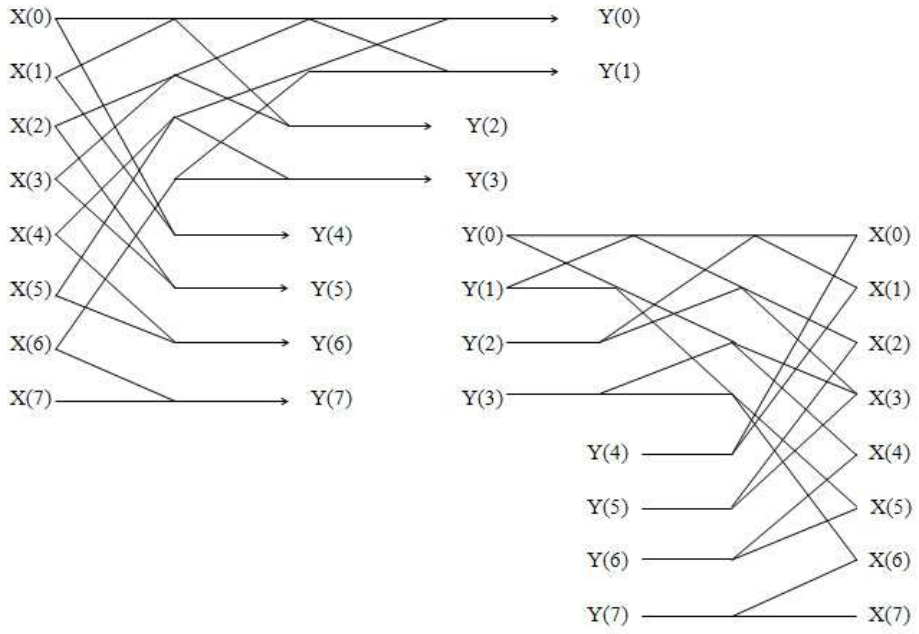
Iš $H(3)$ matricos matome, kad $N/2$ Haar transformacijos koeficientų atitinka gretimų duomenų vektoriaus elementų koreliaciją, tai reiškia, kad skaičiuojant šiuos koeficientus yra įvertinamos tik dviejų gretimų vektoriaus X elementų reikšmės. $N/4$ Haar transformacijos koeficientų – keturių gretimų vektoriaus X elementų reikšmės ir taip toliau, iki N/N koeficientų, kuriuos skaičiuojant yra atsižvelgiama į visas vektoriaus X elementų reikšmes. Taigi, galima teigti, kad HT turi lokališkumo savybę – kuo didesnis HT koeficiento eilės numeris, tuo mažesnį vektoriaus X plotą jis apima.

Apibrėšime atvirkštinę Haar transformaciją (AHT). Kadangi $\mathrm{H}^T(n) \cdot \mathrm{H}(n) = N \cdot \mathrm{E}(n)$, tai (AHT) apibrėžiama tokia lygybe:

$$X = \mathrm{H}^T(n) \cdot Y_X \qquad \textbf{(1.26)}$$

Iš lygybių (1.24) ir (1.26) matyti, kad duomenų vektoriaus X vaizdavimas Haar funkcijomis yra vienareikšmis.

Praktikoje Haar spektro radimui tiesioginis skaičiavimas naudojamas labai retai, nes jis užima per daug laiko. Spektro radimo laikui sutrumpinti yra sudaryti greiti HT algoritmai. Vienas iš jų yra Andrews algoritmas. Šio algoritmo veikimas pagrįstas $H(n)$ matricos struktūros analize. Signaliniai Andrews algoritmo grafai, kai $N = 8$, atrodo šitaip:



(a) **(b)**

1.2 pav. Andrews algoritmas, kai $N = 8$: (a) HT signalinis grafas; (b) AHT signalinis grafas

Remiantis Andrews algoritmo signaliniais grafais (1.2 pav.) ir matricų faktorizavimu, tiesioginę HT transformaciją vektoriui X, kai $N = 8$, galime užrašyti taip:

$$Y_X = \frac{1}{8} H(3) \cdot X = \frac{1}{8} F_1(3) \cdot F_2(3) \cdot F_3(3) \cdot X = \frac{1}{8} \prod_{i=1}^{3} F_i(3) \cdot X \quad \textbf{(1.27)}$$

čia:

$$F_1(3) = diag\left(\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \sqrt{2}, \sqrt{2}, \sqrt{2}, \sqrt{2}, \sqrt{2}, \sqrt{2} \right) \quad \textbf{(1.28)}$$

$$F_2(3) = diag\left(\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix}, \sqrt{2}, \sqrt{2}, \sqrt{2}, \sqrt{2} \right) \quad \textbf{(1.29)}$$

$$F_3(3) = diag\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \quad \textbf{(1.30)}$$

Keturios viršutinės matricos $F_3(3)$ eilutės yra žemo dažnio filtrai, o keturios apatinės – aukšto dažnio filtrai.

Žemiau pateiksime formalizuotą Andrew‘s algoritmo aprašą. Duomenų vektorių X užrašysime šitaip:

$$X = (X(0)X(1)\ \dots\ X(N-1))^T = \left(S^{(0)}(0)\ S^{(0)}(1)\ \dots\ S^{(0)}(2^n-1)\right)^T =$$
$$(O(0)\ E(0)\ O(1)\ E(1)\ \dots\ O(2^{n-1}-1)\ E(2^{n-1}-1))^T \quad \textbf{(1.31)}$$

čia $O(j)$ yra lyginiai duomenų vektoriaus X elementai, o $E(j)$ – nelyginiai; n – iteracijų skaičius, reikalingas atlikti transformacijai.

Atlikę pirmąją iteraciją, gausime tokį vektorių:

$$Y^{(1)} = (S^{(1)}(0),\ S^{(1)}(1), \dots,\ S^{(1)}(2^{n-1}-1),$$
$$D^{(1)}(0),\ D^{(i)}(0),\ D^{(i)}(1), \dots, D^{(i)}(2^{n-i}-1))^T \quad \textbf{(1.32)}$$

čia $S^{(i)}(k) = O^{(i-1)}(k) + E^{(i-1)}(k)$ ir $D^{(i)}(k) = O^{(i-1)}(k) - E^{(i-1)}(k)$. **(1.33; 1.34)**

su visais $k = 0,1, \dots, 2^{n-1}-1$.

Procedūros metu, $S(k)$ reikšmes naudojame kaip tarpinį duomenų vektorių. Po i – tosios ($i \in \{1,2, \dots, n-1\}$) iteracijos vektorius bus toks:

$$Y^{(i)} = (S^{i}(0),\ S^{(i)}(1), \dots,\ S^{(i)}(2^{n-1}-1),$$
$$D^{(i)}(0),\ D^{(i)}(1), \dots, D^{(i)}(2^{n-i}-1))^T \quad \textbf{(1.35)}$$

Atlikę visas n iteracijų, gausime vektorių $Y^{(n)}$. Gautasis vektorius vadinamas Haar spektru ir apibrėžiamas šitaip:

$$Y = (S^{(n)}(0), D^{(n)}(0), D^{(n-1)}(0), D^{(n-1)}(1), D^{(n-2)}(0), \ldots,$$
$$, D_0^{(1)}(0), D^{(1)}(1), \ldots, D^{(1)}(2^{n-1} - 1))^T \quad \textbf{(1.36)}$$

Pastebėkime, kad viršutiniuose skliausteliuose įrašyti skaičiai žymi iteraciją, kurios metu gaunama atitinkamo vektoriaus elemento reikšmė.

1.3.2. DVIMATIS ATVEJIS

Tarkime, turime $N_1 \times N_2$ matmenų matricą $X = [X(m_1, m_2)]$, kur $\forall m_i \in \{0,1, \ldots, N_i - 1\}$, $N_i = 2^{n_i}$, $n_i \in N_i$, $i = 1,2$, tai yra:

$$X = \begin{pmatrix} X(0,0) & X(0,1) & X(0,2) & \ldots & X(0,N_2-1) \\ X(1,0) & X(1,1) & X(1,2) & \ldots & X(1,N_2-1) \\ X(2,0) & X(2,1) & X(2,2) & \ldots & X(2,N_2-1) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ X(N_1-1,0) & X(N_1-1,1) & X(N_1-1,2) & \ldots & X(N_1-1,N_2-1) \end{pmatrix} \quad \textbf{(1.37)}$$

Dvimačiu atveju Haar transformacija (HT) apibrėžiama taip:

$$[Y(k_1, k_2)] = H(n_1) \cdot [X(m_1, m_2)] \cdot H^T(n_2), \quad \textbf{(1.38)}$$

o atvirkštinė Haar transformacja (AHT) šitaip:

$$[X(m_1, m_2)] = H^T(n_1) \cdot [Y(k_1, k_2)] \cdot H(n_2) \quad \textbf{(1.39)}$$

čia $k_i, m_i \in \{0,1, \ldots, N_i - 1\}$; $n_i = \log_2 N_i$; $i = 1,2$.

Matome, kad dvimates transformacijas galima atlikti panaudojant vienmatę transformaciją $N_1 + N_2$ kartų. Kaip minėjome anksčiau, praktikoje Haar spektro radimui tiesioginis skaičiavimas naudojamas labai retai, todėl, kad jį rastume, pasinaudosime Andrews algoritmu (žr. 1.3.1. skyrių).

Pirmiausia šį algoritmą taikysime kiekvienam matricos X stulpeliui (analogiškai, kaip vienmačiu atveju) ir gausime perėjimo matricą $T = [T(l_1, l_2)]$, kur $\forall l_i \in \{0,1, \ldots, N_i - 1\}$, $N_i = 2^{n_i}$, $n_i \in N_i$, $i = 1,2$, tai yra:

$$T = \begin{pmatrix} T(0,0) & T(0,1) & T(0,2) & \dots & T(0,N_2-1) \\ T(1,0) & T(1,1) & T(1,2) & \dots & T(1,N_2-1) \\ T(2,0) & T(2,1) & T(2,2) & \dots & T(2,N_2-1) \\ \dots & \dots & \dots & \dots & \dots \\ T(N_1-1,0) & T(N_1-1,1) & T(N_1-1,2) & \dots & T(N_1-1,N_2-1) \end{pmatrix} \quad \textbf{(1.40)}$$

čia kiekvienas matricos $[T(l_1, l_2)]$ stulpelis yra lygus atitinkamai matricos $[X(m_1, m_2)]$ stulpelio transformacijai (1.36). Toliau Andrews algoritmą taikysime kiekvienai perėjimo matricos T eilutei (analogiškai, kaip vienmačiu atveju) ir gausime Haar spektrą, tai yra:

$$Y = \begin{pmatrix} Y(0,0) & Y(0,1) & Y(0,2) & \dots & Y(0,N_2-1) \\ Y(1,0) & Y(1,1) & Y(1,2) & \dots & Y(1,N_2-1) \\ Y(2,0) & Y(2,1) & Y(2,2) & \dots & Y(2,N_2-1) \\ \dots & \dots & \dots & \dots & \dots \\ Y(N_1-1,0) & Y(N_1-1,1) & Y(N_1-1,2) & \dots & Y(N_1-1,N_2-1) \end{pmatrix} \quad \textbf{(1.41)}$$

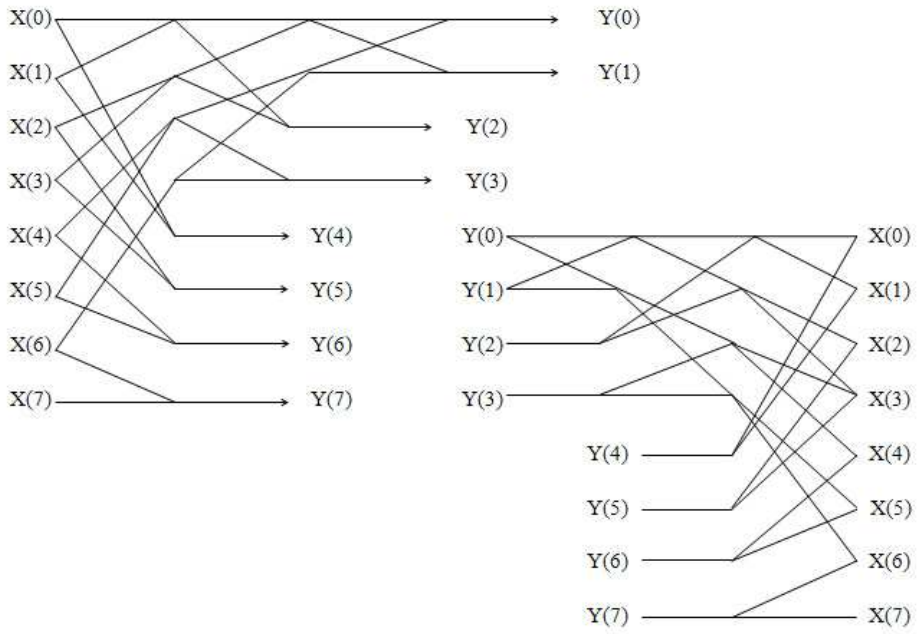
čia kiekviena matricos $[Y(k_1, k_2)]$ eilutė yra lygi atitinkamai matricos $[T(l_1, l_2)]$ eilutės transformacijai (1.36).

Taigi, pritaikius Andrews algoritmą $N_1 + N_2$ kartų (pirmiausia stulpeliams, po to eilutėms), galima atlikti dvimatę Haar transformaciją.

1.4. MODIFIKUOTAS DISKREČIOSIOS HT GREITO APSKAIČIAVIMO ALGORITMAS ANDREWS

1.3.1. skyrelyje aprašytas vienas iš diskrečiajai Haar transformacijai rasti skirtų greito apskaičiavimo algoritmų – Andrews. Nors šio algoritmo dėka Haar spektrą galime rasti greičiau, negu naudodami tiesioginį skaičiavimą, tačiau nepakankamai greitai. Verta pastebėti, kad yra galimybė Andrews algoritmą pagreitinti dar labiau.

Pateikiame signalinius modifikuoto Andrews algoritmo grafus, kai $N = 8$:



(a) **(b)**

1.3 pav. Modifikuotas Andrews algoritmas, kai $N = 8$: (a) HT signalinis grafas; (b) AHT signalinis grafas

Palyginame modifikuoto Andrews algoritmo signalinį grafą su anksčiau pateiktu Andrews algoritmo signaliniu grafu (žr. 1.2 pav.) ir pastebime tokius skirtumus:

1. Modifikuotame Andrews algoritme pašalintas normavimas, tai yra, daugyba iš koeficiento $\sqrt{2}$. Remdamiesi 1.3.1. skyrelyje pateiktais samprotavimais, formules (1.28), (1.29) ir (1.30) galime užrašyti tokiu būdu:

$$F_1(3) = diag\left(\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}, 1,1,1,1,1,1\right) \quad \textbf{(1.42)}$$

$$F_2(3) = diag\left(\begin{pmatrix}1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1\\ 1 & -1 & 0 & 0\\ 0 & 0 & 1 & -1\end{pmatrix}, 1,1,1,1\right) \quad \textbf{(1.43)}$$

$$F_3(3) = diag\begin{pmatrix}1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1\\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1\end{pmatrix} \quad \textbf{(1.44)}$$

Iš formulės (1.27) matyti, kad pasikeitus matricoms $F_i(n)$, kur $i = 1,2,\dots,n$, taip pat keičiasi ir matrica $H(n)$. Pasikeitusios $H(n)$ matricos išraiška yra tokia:

$$H(3) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \begin{matrix} \} \frac{N}{2} \\ \\ \} \frac{N}{4} \\ \\ \\ \\ \\ \end{matrix} \quad \textbf{(1.45)}$$

Kadangi ši matrica yra gauta diskretizuojant Haar funkcijų aibę $\{haar(r,m,x)\}$, tai yra, funkcijas (1.10) ir (1.11), tai šioms funkcijoms taip pat turėtų įsigalioja pakeitimai. Kadangi funkcijos (1.10) reikšmė yra konstanta, tai ši funkcija išlieka nepakitusi, todėl užrašysime tik pakitusią funkciją, tai yra, (1.11):

$$haar(r,m,x) = \begin{cases} 2^{\frac{r}{2}}, & \frac{m-1}{2^r} \le x < \frac{m-\frac{1}{2}}{2^r} \\ -2^{\frac{r}{2}}, & \frac{m-\frac{1}{2}}{2^r} \le x < \frac{m}{2^r} \\ 0, & kitu\ atveju \end{cases} \quad \textbf{(1.46)}$$

Taigi, po normavimo pašalinimo, vienmačiu atveju daugybos veiksmų sumažėja skaičiumi $N-2$ (dviejų pirmųjų spektro koeficientų norma prieš normavimo pašalinimą buvo lygi vienetui). Dvimačiu atveju daugybos veiksmų skaičius sumažėja $(N_1-2)N_2 + (N_2-2)N_1$. Kadangi sumažinome veiksmų skaičių, kurį reikia atlikti Haar transformacijos metu, todėl modifikuotas Andrews algoritmas turėtų veikti greičiau už anksčiau pristatytą Andrews algoritmą (žr. 1.2 pav.).

Pašalinus normavimą, atsiranda ir keletas trūkumų. Vienas iš jų yra tas, kad negalime rasti atvirkštinės diskrečiosios Haar transformacijos naudodami tiesioginį skaičiavimo būdą, tai yra, pagal formulę 1.26. Kitas trūkumas – prarandama diskrečiojo Haar spektro lokalizavimo erdvėje savybė, tai yra, tampa nebeįmanoma iš spektro atkurti pasirinkto vaizdo fragmento (žr. 1.5 skyrelį).

2. Modifikuotame Andrews algoritme pašalinta dalyba iš vektoriaus elementų skaičiaus N. Iš šiame ir 1.3.1. skyreliuose pateiktų samprotavimų išplaukia, kad formulės (1.24) ir (1.27) keičiasi atitinkamai į tokias:

$$Y_X = (Y(0)Y(1) \dots Y(N-1))^T = H(n) \cdot X \quad \textbf{(1.47)}$$

$$Y_X = H(3) \cdot X = F_1(3) \cdot F_2(3) \cdot F_3(3) \cdot X = \prod_{i=1}^{3} F_i(3) \cdot X \quad \textbf{(1.48)}$$

Taigi, po antrosios modifikacijos, vienmačiu atveju dalybos veiksmų sumažėja skaičiumi N, o dvimačiu atveju - skaičiumi $N_1 \times N_2$. Dvimačiu atveju toks veiksmų skaičiaus sumažėjimas gaunamas todėl, kad dalybos veiksmai gali būti atliekami algoritmo pabaigoje, tai yra, kai jau surastas spektras. Kadangi po šios modifikacijos veiksmų, kuriuos reikia atlikti Haar transformacijos metu, skaičių dar labiau sumažinome, tai toks Andrews algoritmas turėtų veikti dar greičiau: tiek lyginant su algoritmu, modifikuotu šio skyrelio 1 dalyje, tiek ir su Andrews algoritmu prieš atliekant modifikavimą (žr. 1.2 pav.).

Nepaisant modifikuoto Andrews algoritmo privalumų ir trūkumų, kurie išvardinti aukščiau šiame skyrelyje, gavome vieną svarbią savybę: Haar spektras, rastas panaudojus šį algoritmą, sudarytas tik iš sveikųjų skaičių. Tariant kitais žodžiais, visi spektro matricos $Y(k_1, k_2)$ elementai priklauso sveikųjų skaičių aibei Z, tai yra, $Y(k_1, k_2) \in Z$ su $\forall k_i \in \{0,1, \dots, N_i - 1\}$, $N_i = 2^{n_i}$, $n_i \in N_i$, $i = 1,2$.

1.5. DISKRETUSIS HAAR SPEKTRAS IR JO SAVYBĖS

Paimkime dvimatę matricą, kurios matmenys yra $N_1 \times N_2$ ir pažymėkime ją $[X(m_1, m_2)]$, kur $\forall m_i \in \{0,1, \dots, N_i - 1\}$, $N_i = 2^{n_i}$, $n_i \in N_i$, $i = 1,2$. Tarkime, kad $[Y(k_1, k_2)]$, kur $\forall k_i \in \{0,1, \dots, N_i - 1\}$, $i = 1,2$ yra vaizdo $[X(m_1, m_2)]$ diskretusis Haar spektras, tai yra, transformacijos metu į ortogonaliąsias funkcijas išskaidytas vaizdas $[X(m_1, m_2)]$.

Kiekvienas spektrinis koeficientas $Y(k_1, k_2)$ turi tokias savybes:

1. Bet kurį k_i galime užrašyti tokiu būdu:

$$k_i = 2^{n_i - s_i} + l_i \quad \textbf{(1.49)}$$

kur $l_i \in \{0,1, \dots, 2^{n_i - s_i} - 1\}$, $i = 1,2$. Spektrinis koeficientas $Y(k_1, k_2)$ yra susijęs su tokiu vaizdo fragmentu:

$$\left[X^{(k_1, k_2)}(m_1, m_2) | (m_1, m_2) \in V_{k_1} \times V_{k_2}\right] \quad \textbf{(1.50)}$$

čia $V_{k_i} = \{l_i \cdot 2^{s_i}, l_i \cdot 2^{s_i} + 1, \dots, (l_i + 1) \cdot 2^{s_i} - 1\}$, $i = 1,2$. Taigi, koeficiento $Y(k_1, k_2)$ reikšmė apibrėžiama vaizdo fragmentu X_{k_1,k_2}, o tai yra lokalizavimas erdvėje.

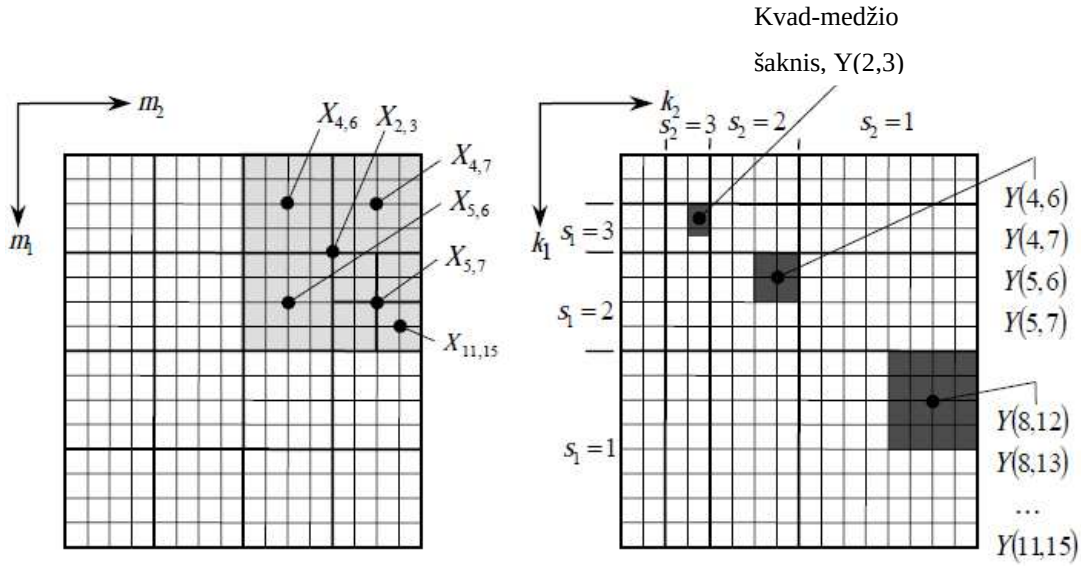
2. Jeigu $s_1 > 1$ ir $s_2 > 1$, tada koeficientui $Y(k_1, k_2)$ galima priskirti kvad-medį, kurio spektriniai koeficientai (viršūnės) aprašomi tokia aibe:

$$\left\{Y(k_1^*, k_2^*) \middle| (k_1^*, k_2^*) \in \cup_{t=1}^{min\{s_1-1, s_2-1\}} (\tau_{k_1}(t) \times \tau_{k_2}(t))\right\} \quad \textbf{(1.51)}$$

čia $\tau_{k_i}(t) = \{2^t \cdot k_i\ , 2^t \cdot k_i + 1, \dots, 2^t(k_i + 1) - 1\}, \quad t = 1, 2, \dots, s_i - 1\ , i = 1,2$ **(1.52)**

Koeficientas $Y(k_1, k_2)$ vadinamas kvad-medžio šaknimi.

Pateiksime grafinę šių savybių interpretaciją, kai $N_1 = N_2 = 16$



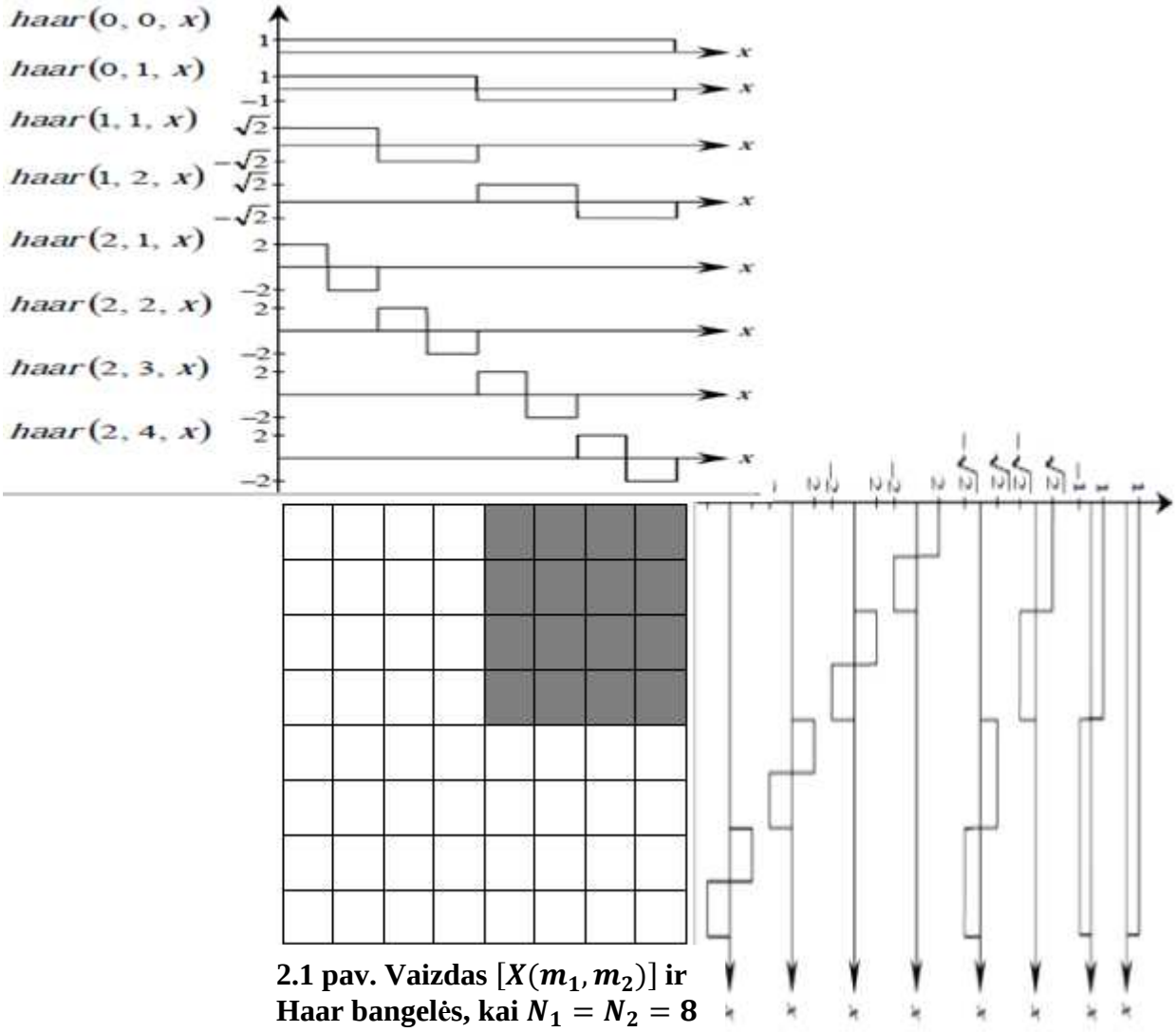
(a) (b)

1.4 pav. Grafinė spektrinių koeficientų interpretacija: (a) vaizdas $[X(m_1, m_2)]$; koeficientas (kvad-medžio šaknis) $Y(2,3)$ yra susijęs su vaizdo fragmentu $X_{2,3}$; (b) spektras $[Y(k_1, k_2)]$; koeficientui $Y(2,3)$ priskirtas kvad-medis ($\tau_2 = \tau_2(1) \cup \tau_2(2) = \{4,5,8,9,10,11\}$ ir $\tau_3 = \tau_3(1) \cup \tau_3(2) = \{6,7,12,13,14,15\}$)

2. GREITO HT APSKAIČIAVIMO VAIZDO FRAGMENTAMS PROCEDŪROS IR JŲ ANALIZĖ

2.1. VAIZDO FRAGMENTŲ IDENTIFIKAVIMAS

Turime Haar funkcijų aibę $\{har(r_i, m_i, x)\}$, kur $x \in [0,1)$, $i = 1, 2$, tai yra, funkcijas (1.10) ir (1.11). Pavaizduosime, kaip iš skaitmeninio vaizdo $[X(m_1, m_2)]$ išrenkamas jo fragmentas $\left[X^{(k_1,k_2)}(m_1, m_2)\right]$.



2.1 pav. Vaizdas $[X(m_1, m_2)]$ ir Haar bangelės, kai $N_1 = N_2 = 8$

Paimkime aibę $c_i = \{0, 1, \dots, N_i - 1\}$, $i = 1, 2$. Kiekvienai Haar bangelei, pradedant nuo $har(0,0,x)$, priskirkime aibės c_i elementą tokiu būdu:

$har(0,0,x)$ yra pirmasis aibės c_i elementas;

$har(0,1,x)$ yra antrasis aibės c_i elementas;

$har(r_i, m_i, x)$ yra $(N_i - 1)$-tasis aibės c_i elementas.

Kai $c_1 = 2$ ir $c_2 = 3$, gausime vaizdo fragmentą $\left[X^{(2,3)}(m_1, m_2)\right]$ (žr. 2.1 pav.), fragmentas paryškintas). Taigi, $\left[X^{(k_1,k_2)}(m_1, m_2)\right] = \left[X^{(c_1,c_2)}(m_1, m_2)\right]$.

2.2. REKURENTINIS FRAGMENTINIO HT SPEKTRO RADIMO ALGORITMAS

Paimkime spektrinį koeficientą $Y(k_1, k_2)$, kur $k_i = 2^{n_i - s_i} + l_i$, $i = 1,2$. Koeficientas $Y(k_1, k_2)$ yra susijęs su vaizdo fragmentu $\left[X^{(k_1,k_2)}(m_1, m_2)\right]$. Šio vaizdo fragmento spektrą pažymėkime $\left[Y^{(k_1,k_2)}(u, v)\right]$, kur $u = 0,1, \dots, 2^{s_1 - 1}$ ir $v = 0,1, \dots, 2^{s_2 - 1}$. Haar spektro savybės leido sudaryti ir realizuoti efektyvią perėjimo nuo Haar spektro visam vaizdui prie Haar spektro vaizdo fragmentui procedūrą.

1. Spektrinis koeficientas $Y^{(k_1,k_2)}(0,0)$, kuris yra susijęs su vaizdo fragmento X_{k_1,k_2} pastoviąja dedamąja $\bar{X}_{k_1,k_2}$, apskaičiuojamas naudojant rekurentinius ryšius:

$$Y^{(k_1,k_2)}(0,0) = \bar{X}_{k_1,k_2} = \bar{X}_{k_1'.k_2'} + (-1)^{k_1} \cdot \sqrt{2^{n_1 - s_1 - 1}} \cdot A(k_1', k_2') + (-1)^{k_2} \cdot \sqrt{2^{n_2 - s_2 - 1}} \cdot B(k_1', k_2') + (-1)^{k_1 + k_2} \cdot \sqrt{2^{n_1 + n_2 - s_1 - s_2 - 2}} \cdot Y(k_1', k_2') \quad \textbf{(2.1)}$$

kol $k_1 > 1$ ir $k_2 > 1$. Čia $k_i' = [k_i/2]$ yra sveikoji skaičiaus $k_i/2$ dalis, $i = 1,2$.

Atskiri atvejai:

$$Y^{(0,k_2)}(0,0) = \bar{X}_{0,k_2} = \bar{X}_{1,k_2} = \bar{X}_{0.k_2'} + (-1)^{k_2} \cdot \sqrt{2^{n_2 - s_2 - 1}} \cdot B(0, k_2'), \quad \textbf{(2.2)}$$

kai $k_1 = 1$, o $k_2 > 1$

$$Y^{(k_1,0)}(0,0) = \bar{X}_{k_1,0} = \bar{X}_{k_1,1} = \bar{X}_{k_1',0} + (-1)^{k_1} \cdot \sqrt{2^{n_1 - s_1 - 1}} \cdot A(k_1', 0), \quad \textbf{(2.3)}$$

kai $k_1 > 1$, o $k_2 = 1$

$$A(k_1, k_2) = \begin{cases} Y(k_1, 0), kai\ k_2 \le 1\ (s_2 = n_2) \\ Y(k_1, 0) + \sum_{r=1}^{n_2 - s_2} (-1)^{\beta_{r-1}} \cdot \sqrt{2^{n_2 - s_2 - r}} \cdot Y(k_1, \beta_r), kai\ k_2 > 1\ (s_2 < n_2) \end{cases} \quad \textbf{(2.4)}$$

$$B(k_1,k_2)=$$

$$\begin{cases} Y(0,k_2), kai\ k_1 \le 1\ (s_1=n_1) \\ Y(0,k_2)+\sum_{r=1}^{n_1-s_1}(-1)^{\alpha_{r-1}}\cdot\sqrt{2^{n_1-s_1-r}}\cdot Y(\alpha_r,k_2), kai\ k_1>1\ (s_1<n_1) \end{cases} \quad \textbf{(2.5)}$$

čia $\alpha_r=[\alpha_{r-1}/2]$, kur $r=1,2,\dots,n_1-s_1$, o $\alpha_0=k_1$; $\beta_r=[\beta_{r-1}/2]$, kur $r=1,2,\dots,n_2-s_2$, o $\beta_0=k_2$;

Taip pat pastebėkime, kad $\bar{X}_{0,0}=\bar{X}_{0,1}=\bar{X}_{1,0}=\bar{X}_{1,1}$ **(2.6)**

2. Spektriniai koeficientai $Y^{(k_1,k_2)}(u,v)$, kai $u^2+v^2\neq 0$, randami naudojant šias formules:

$$Y^{(k_1,k_2)}(u,0)=\sqrt{2^{n_1-s_1}}\cdot A(k_1',k_2) \quad \textbf{(2.7)}$$

$$Y^{(k_1,k_2)}(0,v)=\sqrt{2^{n_2-s_2}}\cdot B(k_1,k_2') \quad \textbf{(2.8)}$$

$$Y^{(k_1,k_2)}(u,v)=\sqrt{2^{n_1+n_2-s_1-s_2}}\cdot Y(k_1',k_2') \quad \textbf{(2.9)}$$

su visais $u=1,2,\dots,2^{s_1}-1$ ir $v=1,2,\dots,2^{s_2}-1$; indeksai k_1' ir k_2' prilyginami atitinkamai u-tajam ir v-tajam sutvarkytų (didėjančia tvarka) aibių $\{k_1\}\cup\tau_{k_1}$ ir $\{k_2\}\cup\tau_{k_2}$ elementams. Čia $\tau_{k_i}=\tau_{k_i}(1)\cup\tau_{k_i}(2)\cup\dots\cup\tau_{k_i}(s_i-1)$, $i=1,2$.

2.3. MODIFIKUOTA VERSIJA

2.2 skyrelyje aptarta procedūra iš pirmo žvilgsnio atrodo gana sudėtingai. Pertvarkysime šią procedūrą į lengviau suprantamą. Tam įvesime tokius žymėjimus:

$[X(m_1,m_2)]$ – dvimatis skaitmeninis vaizdas, kur $\forall m_r\in\{0,1,\dots,N_r-1\}$, $N_r=2^{n_r}$, $n_r\in N_r$, $r=1,2$.

$[Y(k_1,k_2)]$ - dvimatis diskretusis Haar skaitmeninio vaizdo $[X(m_1,m_2)]$ spektras, kur $\forall k_r\in\{0,1,\dots,N_r-1\}$, $N_r=2^r$, $n_r\in N_r$, $r=1,2$; kai $k_r\neq 0$, galime užrašyti tokiu būdu: $k_r=2^{n_r-i_r}+j_r$, $i_r\in\{1,2,\dots,n_r\}$, $j_r\in\{0,1,\dots,2^{n_r-i_r}-1\}$

$\left[X^{(k_1,k_2)}(m_1,m_2)|(m_1,m_2)\in V_{k_1}\times V_{k_2}\right]$, kur $V_{k_r}=\left\{j_r\cdot 2^{i_r}, j_i\cdot 2^{i_r}+1,\dots,(j_r+1)\cdot 2^{i_r}-1\right\}$, $r=1,2$, yra vaizdo $[X(m_1,m_2)]$ fragmentas $2^{i_1}\times 2^{i_2}$, susijęs su spektriniu koeficientu $Y(k_1,k_2)$.

$\left[Y^{(k_1,k_2)}(u,v)\right]$, kur $u=0,1,\dots,2^{i_1-1}$ ir $=0,1,\dots,2^{i_2-1}$, yra diskretusis vaizdo fragmento $\left[X^{(k_1,k_2)}(m_1,m_2)\right]$ Haar spektras.

Pateikiame pertvarkytą procedūrą:

1. Pirmiausia formuojame aibes (kai $k_1>1$ ir $k_2>1$)

$$S_V=\left\{\alpha_0,\ \alpha_1,\dots,\alpha_{n_1-i_1}\right\}, \quad \textbf{(2.10)}$$

kur $\alpha_0=k_1$, $\alpha_s=[\alpha_{s-1}/2]$, $s=1,2,\dots,n_1-i_1$

$$S_H=\left\{\beta_0,\beta_1,\dots,\beta_{n_2-i_2}\right\}, \quad \textbf{(2.11)}$$

kur $\beta_0=k_2$, $\beta_t=[\beta_{t-1}/2]$, $t=1,2,\dots,n_2-i_2$

$$\tau_{k_r}=\{k_r\}\cup\left\{\bigcup_{q=1}^{i_r-1}\tau_{k_r}(q)\right\}, \quad \textbf{(2.12)}$$

kur $\tau_{k_r}(q)=\{2^q k_r,\ 2^q k_r+1,\dots,2^q(k_r+1)-1\}$, $r=1,2$; be to, aibė τ_{k_r} yra sutvarkyta (elementų didėjimo tvarka).

2. Apskaičiuojame spektrinius koeficientus vaizdo fragmentui $\left[X^{(k_1,k_2)}(m_1,m_2)\right]$:

$$Y^{(k_1,k_2)}(0,0)=Y(0,0)+\sum_{s=1}^{n_1-i_1}(-1)^{\alpha_{s-1}}\cdot\sqrt{2^{n_1-i_1-s}}\cdot Y(\alpha_s,0)+\sum_{t=1}^{n_2-i_2}(-1)^{\beta_{t-1}}\cdot\sqrt{2^{n_2-i_2-t}}\cdot Y(0,\beta_t)+\sum_{t=1}^{n_2-i_2}\sum_{s=1}^{n_1-i_1}(-1)^{\alpha_{s-1}+\beta_{t-1}}\cdot\sqrt{2^{n_1+n_2-i_1-i_2-s-t}}\cdot Y(\alpha_s,\beta_t) \quad \textbf{(2.13)}$$

$$Y^{(k_1,k_2)}(u,0)=\sqrt{2^{n_1-i_1}}\left(Y(k_1',0)+\sum_{t=1}^{n_2-i_2}(-1)^{\beta_{t-1}}\cdot\sqrt{2^{n_2-i_2-t}}\cdot Y(k_1',\beta_t)\right) \quad \textbf{(2.14)}$$

su visais $u=1,2,\dots,2^{i_1}-1$; k_1' yra u-tasis aibės τ_{k_1}elementas (elementų numeracija aibėje τ_{k_1} pradedama nuo vieneto).

$$Y^{(k_1,k_2)}(0,v)=\sqrt{2^{n_2-i_2}}\left(Y(0,k_2')+\sum_{s=1}^{n_1-i_1}(-1)^{\alpha_{s-1}}\cdot\sqrt{2^{n_1-i_1-s}}\cdot Y(\alpha_s,k_2')\right) \quad \textbf{(2.15)}$$

su visais $v=1,2,\dots,2^{i_2}-1$; k_2' yra v-tasis aibės τ_{k_2}elementas (elementų numeracija aibėje τ_{k_2} pradedama nuo vieneto).

$$Y^{(k_1,k_2)}(u,v) = \sqrt{2^{n_1+n_2-i_1-i_2}} \cdot Y(k_1', k_2') \quad \textbf{(2.16)}$$

su visais $u = 1, 2, \dots, 2^{i_1} - 1$ ir $v = 1, 2, \dots, 2^{i_2} - 1$; k_1' ir k_2' yra atitinkamai u-tasis ir v-tasis aibių τ_{k_1} ir τ_{k_2} elemetai (elementų numeracija aibėse τ_{k_1} ir τ_{k_2} pradedama nuo vieneto).

Atskiri atvejai:

1. Kai $k_1 = 1$, o $k_2 > 1$, turime:

$$Y^{(1,k_2)}(u,0) = Y(u,0) + \sum_{t=1}^{n_2-i_2} (-1)^{\beta_{t-1}} \cdot \sqrt{2^{n_2-i_2-t}} \cdot Y(u, \beta_t), \quad \textbf{(2.17)}$$

su visais $u = 0, 1, \dots, 2^{n_1} - 1$

$$Y^{(1,k_2)}(u,v) = \sqrt{2^{n_2-i_2}} \cdot Y(u, k_2'), \quad \textbf{(2.18)}$$

su visais $u = 0, 1, \dots, 2^{n_1} - 1$ ir $v = 1, 2, \dots, 2^{i_2} - 1$;

čia k_2' yra v-tasis aibės τ_{k_2}elementas (elementų numeracija aibėje τ_{k_2} pradedama nuo vieneto).

2. Kai $k_1 > 1$, o $k_2 = 1$ turime:

$$Y^{(k_1,1)}(0,v) = Y(0,v) + \sum_{s=1}^{n_1-i_1} (-1)^{\alpha_{s-1}} \cdot \sqrt{2^{n_1-i_1-s}} \cdot Y(\alpha_s, v), \quad \textbf{(2.19)}$$

su visais $v = 0, 1, \dots, 2^{n_2} - 1$

$$Y^{(k_1,1)}(u,v) = \sqrt{2^{n_1-i_1}} \cdot Y(k_1', v), \quad \textbf{(2.20)}$$

su visais $u = 1, 2, \dots, 2^{i_1} - 1$ ir $v = 0, 1, \dots, 2^{n_2} - 1$;

čia k_1' yra u-tasis aibės τ_{k_1}elementas (elementų numeracija aibėje τ_{k_1} pradedama nuo vieneto).

3. VAIZDO SUSPAUDIMO SPEKTRINĖJE HT SRITYJE METODAS SPIHT

Vaizdo suspaudimo metodų atsiradimą įtakojo poreikis apriboti grafinės informacijos kiekį vaizduose tam, kad be reikalo nebūtų eikvojami informacinių technologijų resursai bei

informacijos perdavimo greitis išliktų kiek įmanoma didesnis. Egzistuoja daugybė diskrečiosiomis bangelių transformacijomis paremtų vaizdo suspaudimo metodų, kuriais pasinaudoję galime suspausti grafinę informaciją ir ją atkurti iki nustatyto detalumo lygmens. Kaip pavyzdys, tokie algoritmai yra: EZW (Embedded Zerotree Wavelet), EM (Expectation Maximization), EBCOT (Embedded Block Coding with Optimal Truncation), SPIHT (Set Partitioning in Hierarchical Trees), STW (Spatial-orientation Tree Wavelet), WDR (Wavelet Diference Reduction) ir kiti.

Darbe vaizdo suspaudimą atliksime SPIHT metodo pagalba. Šis metodas paremtas EZW procedūros veikimo principais, tačiau yra labiau racionalizuotas. Pirmąją EZW procedūrą 1993 metais pasiūlė J. M. Shapiro. Tuo metu EZW procedūra grafinės informacijos suspaudimo efektyvumu ir greitumu pranoko visus tuometinius metodus, skirtus suspausti vaizdui. Pasinaudoję EZW metodo veikimo principu, 1995 metais Amir Said ir William A. Pearlman pasiūlė pirmąją SPIHT metodo versiją, kuri grafinės informacijos suspaudimo efektyvumu bei greitumu pranoko EZW procedūrą. SPIHT algoritmą įvairūs mokslininkai tobulina iki dabar, todėl net ir šiomis dienomis vaizdui suspausti skirtas SPIHT metodas išlieka vienu iš greičiausių bei efektyviausių.

3.1. PROGRESYVIOJI INFORMACIJOS PERDAVIMO SCHEMA

Šiame skyrelyje nagrinėjamos progresyviosios informacijos perdavimo schemos pagrindinis tikslas yra išrinkti iš vaizdo ir iš jo spektro, gauto diskrečiosios bangelių transformacijos metu, svarbiausią informaciją bei tokią informaciją perduoti pirmiausia. Rašydami ,,svarbiausia informacija“ omenyje turime didžiausias vaizdo ir spektro matricų elementų reikšmes, kadangi kuo didesnės reikšmės perduodamos, tuo atitinkamai labiau mažėja vaizdo ir spektro iškraipymai. Aprašytam informacijos išrinkimo iš vaizdo iškraipymui nustatyti naudojame vidutinę kvadratinę paklaidą, tai yra:

$$DV_{VKP}\left(X - \tilde{X}\right) = \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left(X(i,j) - \tilde{X}(i,j)\right)^2 \quad \textbf{(3.1)}$$

čia X – pradinio vaizdo matrica; $\tilde{X}$ - suspausto vaizdo matrica; M ir N - atitinkamai matricos eilučių ir stulpelių skaičius (pradinės vaizdo matricos ir suspausto vaizdo matricos matmenys tokie patys); $X(i,j)$ - pradinės vaizdo matricos i– tosios eilutės ir j - tojo stulpelio elementas; $\tilde{X}(i,j)$ - suspausto vaizdo matricos i– tosios eilutės ir j - tojo stulpelio elementas.

Analogiškai užrašome formulę informacijos išrinkimo iš spektro iškraipymui nustatyti:

$$DS_{VKP}\left(Y - \tilde{Y}\right) = \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left(Y(i,j) - \tilde{Y}(i,j)\right)^2 \quad \textbf{(3.2)}$$

čia Y – pradinio spektro matrica; $\tilde{Y}$ – suspausta spektro matrica; $Y(i,j)$ - pradinės spektro matricos i– tosios eilutės ir j - tojo stulpelio elementas; $\tilde{Y}(i,j)$ - suspausto spektro matricos i– tosios eilutės ir j - tojo stulpelio elementas; kiti formulėje naudojami žymėjimai yra tokie patys, kaip formulėje (3.1).

Iš formulės (3.2) matome, kad perdavus spektro elementą $Y(i,j)$ dekodavimo algoritmui (paprasčiau tariant, perkėlus elementą iš matricos Y į matricą $\tilde{Y}$), vidutinė kvadratinė paklaida sumažės skaičiumi $\frac{(Y(i,j))^2}{M \cdot N}$. Tai reiškia, kad spektro Y didžiausi elementai dekodavimo algoritmui turėtų būti perduodami pirmiausia, nes jie perneša didžiausią informacijos kiekį.

3.2. DVEJETAINE FORMA IŠREIKŠTŲ ELEMENTŲ PERDAVIMAS

Bet kuris sveikasis spektro elementas $Y(i,j)$ gali būti konvertuotas į dvejetainę formą tokiu būdu:

$$Y(i,j) = \sum_{n=0} Y_n(i,j) \cdot 2^n \quad \textbf{(3.3)}$$

Tuomet remdamiesi 3.1 skyrelyje aprašyta progresyviąją informacijos perdavimo schema galime įžvelgti, kad spektro elemento, konvertuoto į dvejetainę išraišką, reikšmingiausi bitai turėtų būti perduodami pirmiausia. Reikšmingiausias bitas yra toks, kuris gautas iš formulės (3.3) prie didžiausios n reikšmės, o mažiausiai reikšmingas bitas – gautas prie mažiausios n reikšmės, tai yra, nulio.

Atsižvelgdami į anksčiau pateiktus samprotavimus galime teigti, kad visi spektro Y elementai turi būti išdėstomi mažėjimo tvarka ir pirmiausia perduodami šių elementų reikšmingiausi bitai. Taip pat akivaizdu, kad kuo spektro elementas didesnis, tuo daugiau bitų panaudojama jį užrašyti dvejetaine forma. Svarbiausios informacijos, išreikštos dvejetaine forma, tai yra, bitais, perdavimo eilės tvarką pateikiame lentelėje:

3.1 lentelė

Svarbiausios informacijos, išreikštos bitais, perdavimo eilės tvarka

n	s	s	s	s	s	s	s	s
3	1	1	0	0	0	0	0	0
2	→	→	1	1	0	0	0	0
1	→	→	→	→	1	1	1	1
0	→	→	→	→	→	→	→	→

Pirmąjame šios lentelės stulpelyje yra spektro elemento $Y(i,j)$, išreikšto dvejetaine forma, n - tasis bitas, o pirmojoje eilutėje – šio elemento ženklas, tai yra, plius arba minus. Sekančiuose šios lentelės stulpeliuose (išskyrus pirmąjį) yra dvejetaine forma išreikšti spektro Y elementai, kurie, žiūrint iš kairės į dešinę, išdėstyti mažėjimo tvarka. Nesunku pastebėti, kad reikšmingiausi bitai yra viršutinėje lentelės eilutėje, sekančioje – mažiau reikšmingi, o apatinėje – mažiausiai reikšmingi. Taigi, spektro Y elementai pagal dydį yra surikiuoti nuo didžiausio iki mažiausio ir kiekvieno elemento bitai pagal reikšmingumą išdėstyti mažėjimo tvarka.

Įsivaizduokime, kad be informacijos perdavimo eilės tvarkos, dekodavimo algoritmui papildomai perduodame koeficientus μ_n, kur μ_n yra atitinkamai kiekvienoje 3.1 lentelės eilutėje esančių bitų, lygių 1, skaičius. Pateiktoje lentelėje μ_n reikšmės yra tokios: $\mu_3 = 2$, $\mu_2 = 2$, $\mu_1 = 4$ ir taip toliau. Pastebėkime, kad kai dekodavimo algoritmui prie tam tikros n reikšmės perduodamas pirmasis spektro elemento $Y(i,j)$ bitas, lygus 1, tai spektro Y elementas patenka į tokį intervalą:

$$2^n \leq |Y(i,j)| < 2^{n+1} \qquad \textbf{(3.4)}$$

Dekodavimo algoritmui pirmiausia perduodami reikšmingiausi bitai, tai yra, pradedant viršutiniąją 3.1 lentelės eilute (informacijos perdavimo eiliškumas pavaizduotas rodyklėmis). Kadangi kiekviename šios lentelės stulpelyje bitai išdėstyti reikšmingumo mažėjimo tvarka, todėl pirmaujančiųjų „0“ bitų ir pirmųjų „1“ bitų dekodavimo algoritmui perduoti nereikia. Tokius bitus galime nesunkiai nuspėti iš koeficiento μ_n reikšmės ir atsižvelgę į informacijos perdavimo eilės tvarką.

3.3. ELEMENTŲ REIKŠMINGUMO NUSTATYMAS IR RŪŠIAVIMO Į AIBES ALGORITMAS

Pagrindinis šio algoritmo principas yra išrinkti ne visus mažėjimo tvarka surikiuotus spektro Y elementus, o tik tokius, kurie priklausytų intervalui $2^n \leq |Y(i,j)| < 2^{n+1}$, pradedant maksimalia n reikšme ir ją mažinant vienetu po kiekvieno žingsnio. Jeigu n - tąjame žingsnyje spektro elementas $|Y(i,j)| \geq 2^n$, tai jis vadinamas reikšmingu, o priešingu atveju – nereikšmingu.

Algoritmas spektro Y elementus, kurių koordinatės saugomos aibėje Γ, surūšiuoja į šios aibės poaibius Γ_m (čia m yra m – tasis aibės Γ elementas) ir kiekviename poaibyje pagal formulę

$$\max_{(i,j)\in\Gamma_m}\{|Y(i,j)|\} \geq 2^n \qquad \textbf{(3.5)}$$

nustato elementų $Y(i,j)$ reikšmingumą. Jeigu nelygybė netenkinama, vadinasi poaibis Γ_m yra nereikšmingas, tai yra, visi elementai, esantys šiame poaibyje, yra nereikšmingi. Jeigu nelygybė tenkinama, vadinasi poaibis Γ_m yra reikšmingas ir tokiu atveju jis skirstomas į naujus poaibius $\Gamma_{m,l}$, kurių elementams reikšmingumas taip pat nustatomas pagal formulę 3.5. Elementų reikšmingumas nustatinėjamas tol, kol surandami visi reikšmingi spektro Y elementai. Jų skirstymas į poaibius remiasi erdvėje orientuotų medžų savybėmis.

Pateikiame formulę, kuri naudojama užkodavimo ir dekodavimo algoritmuose spektro Y elementų reikšmingumui nustatyti:

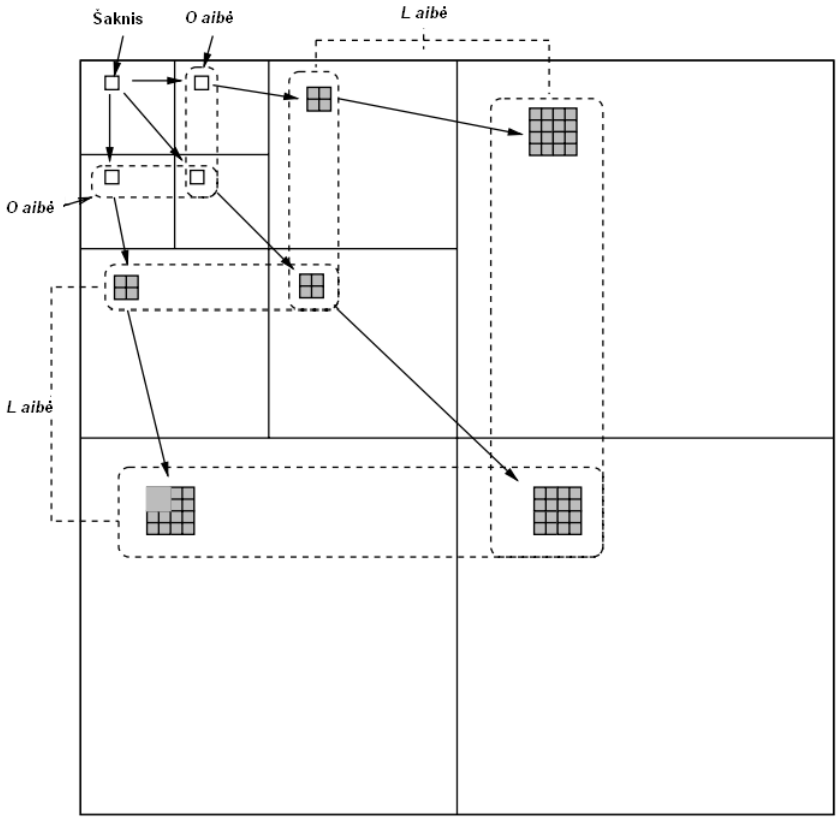
$$S_n(\Gamma) = \begin{cases} 1, \max_{(i,j)\in\Gamma_m}\{|Y(i,j)|\} \geq 2^n \\ 0, \text{kitu atveju} \end{cases} \qquad \textbf{(3.6)}$$

3.4. ERDVĖJE ORIENTUOTI MEDŽIAI

Pasinaudodami 1.5 skyrelyje aprašyta antrąja savybe parodome, kaip erdvėje orientuoto medžio elemetai skirstomi į aibes. Pirmiausia, apibrėžiame keturias aibes:

$H(i,j)$ - elemento $Y(i,j)$ šaknų (tėvų) aibė, išskyrus $Y(0,0)$
$O(i,j)$ – elemento $Y(i,j)$ tiesioginių palikuonių (vaikų) aibė
$D(i,j)$ – elemento $Y(i,j)$ visų palikuonių (vaikų, anūkų, proanūkių ir taip toliau) aibė
$L(i,j)$ – elemento $Y(i,j)$ visų palikuonių, išskyrus tiesioginius (vaikus), aibė

Nesunku pastebėti, kad $L(i,j) = D(i,j) - O(i,j)$. Pateikiame grafinę erdvėje orientuoto medžio interpretaciją:



3.1 pav. Erdvėje orientuotas medis

Kiekvienas šiame medyje esantis elementas turi tiesioginius palikuonius, kurių aibė formuojama šitaip:

$$O(i,j) = \{(2i;2j),(2i;2j+1),(2i+1;2j),(2i+1;2j+1)\} \quad \textbf{(3.7)}$$

Tokias erdvėje orientuoto medžio dalis naudojame rūšiavimo į aibes algoritme ir nuo jų priklauso spektro Y elementų skirstymas į poaibius, kuris vadovaujasi šiomis taisyklėmis:

1. Iš pradžių formuojamos aibės $\{(i,j)\}$ ir $D(i,j)$, kur $\forall(i,j) \in H$.
2. Jeigu $D(i,j)$ elementas yra reikšmingas, tai ši aibė suskirstoma į aibę $L(i,j)$ ir į keturių elementų aibę $\{(k,l)\}$, kur $\forall(k,l) \in O(i,j)$.
3. Jeigu $L(i,j)$ yra reikšminga, tai ji suskirstoma į keturias aibes $D(k,l)$, kur $\forall(k,l) \in O(i,j)$.

3.5. UŽKODAVIMO ALGORITMAS

Nustatinėjant aibėse esančių elementų reikšmingumą, jų išdėstymo tvarka šiose aibėse yra svarbi, todėl informacija apie šių elementų reikšmingumą yra saugoma trijuose sąrašuose: LIS – nereikšmingų aibių sąrašas, LIP – nereikšmingų pikselių sąrašas ir LSP – reikšmingų

pikselių sąrašas. Visuose trijuose sąrašuose kiekvienas elementas yra spektro elemento Y koordinatės (i, j). LIP ir LSP sąrašuose šiomis koordinatėmis nusakomas vienas pikselis, o LIS sąraše – aibė $D(i, j)$ arba aibė $L(i, j)$. Kad galėtume atskirti, kurios aibės koordinatės nusakomos, tai kiekvienam sąrašo LIS elementui priskiriame du tipus: A arba B. Čia tipas A reiškia, kad sąrašo elementas nusakomas aibės $D(i, j)$ koordinatėmis, o tipas B – aibės $L(i, j)$ koordinatėmis.

Pateikiame vaizdo užkodavimui atlikti reikalingus algoritmo žingsnius:

1. Iniciacija
   1.1. Randame maksimalų spektro elementą, tai yra: $n = [log_2(\max|Y(i,j)|)]$, čia laužtiniai skliausteliai reiškia dešinėje lygybės pusėje gauto rezultato sveikąją dalį
   1.2. Sąrašą LSP paliekame tuščią, tai yra: $LSP = \{\emptyset\}$
   1.3. Sąrašą LIP užpildome aibės $\mathcal{H}$ elementais, tai yra: $LIP = \{(0;0), (0;1), (1;0), (1;1)\}$
   1.4 Sąrašą LIS užpildome aibės $\mathcal{H}$ elementais, kurie turi palikuonių, ir nustatome tipą A, tai yra: $LIS = \{(0;1)A, (1;0)A, (1;1)A\}$
2. Rūšiavimo vykdymas
   2.1. Su kiekvienu sąrašo LIP elementu atliekame veiksmus
      2.1.1. Išvedame $S_n(i, j)$
      2.1.2. Jeigu $S_n(i, j) = 1$, tai perkeliame elementą (i, j) į sąrašą LSP ir išvedame spektro elemento $Y(i, j)$ ženklą.
   2.2. Su kiekvienu sąrašo LIS elementu atliekame tokius veiksmus
      2.2.1. Jeigu elemento tipas sąraše yra A, tai
         2.2.1.1. Išvedame $S_n(D(i, j))$
         2.2.1.2. Jeigu $S_n(D(i, j)) = 1$, tai
            2.2.1.2.1. Su kiekvienu $(k, l) \in O(i, j)$ atliekame tokius veiksmus
               2.2.1.2.1.1. Išvedame $S_n(k, l)$
               2.2.1.2.1.2. Jeigu $S_n(k, l) = 1$, tai pridedame elementą (k, l) prie sąrašo LSP ir išvedame spektro elemento $Y(k, l)$ ženklą
               2.2.1.2.1.3. Jeigu $S_n(k, l) = 0$, tai pridedame elementą (k, l) prie sąrašo LIP
            2.2.1.2.2. Jeigu $L(i, j) \neq \{\emptyset\}$, tai perkeliame (i, j) į sąrašą LIS su tipu B ir vykdome žingnelį 2.2.2.
            2.2.1.2.3. Jeigu $L(i, j) = \{\emptyset\}$, tai pašaliname (i, j) iš sąrašo LIS
      2.2.2. Jeigu elemento tipas sąraše yra B, tai
         2.2.2.1. Išvedame $S_n(L(i, j))$

2.2.2.2. Jeigu $S_n(L(i,j)) = 1$, tai

2.2.2.2.1. Pridėti kiekvieną $(k,l) \in O(i,j)$ prie sąrašo LIS su tipu A

2.2.2.2.2. Pašalinti (i,j) iš sąrašo LIS

3. Tobulinimo vykdymas

3.1. Kiekvienam (i,j) iš sąrašo LSP, išskyrus tuos elementus, kurie į šį sąrašą pateko paskutinio rūšiavimo vykdymo metu, tai yra, su ta pačia n reikšme, išvedame elemento $Y(i,j)$ n - tąjį reikšmingiausią bitą.

4. Kvantavimo žingsnio atnaujinimas

4.1. Sumažiname n skaičiumi 1 ir vykdome 2 – ajį algoritmo žingsnį, tai yra, rūšiavimą

Pastaba: elementai visada dedami į sąrašų LIS, LIP, LSP pabaigas neatsižvelgiant į tai, ar jie perkeliami, ar tik pridedami prie atitinkamo sąrašo.

3.6. DEKODAVIMO ALGORITMAS

Dekodavimo algoritmas yra analogiškas užkodavimo algoritmui, todėl 3.1. skyrelyje aprašytame algoritme žodį ,,išvesti" reikėtų pakeisti į žodį ,,įvesti". Tiesa, lyginant su užkodavimo algoritmu, dekodavimo algoritmas atlieka kelis papildomus uždavinius.

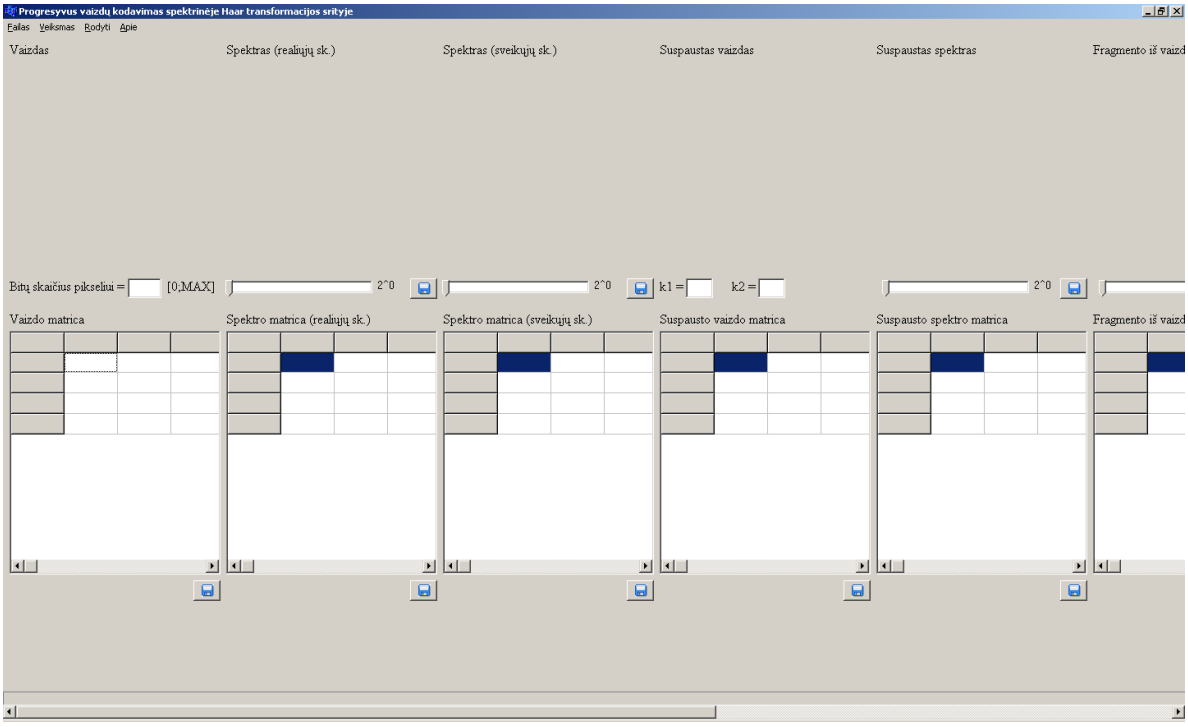
Pirmasis jų atliekamas tada, kai prie tam tikros n reikšmės koordinatė perkeliama į LSP sąrašą. Iš 3.2 skyrelyje gautų samprotavimų žinome, kad elementas, kuris aprašomas tokia koordinate, patenka į intervalą (3.4). Taigi, dekoderis, naudodamas šią informaciją ir elemento ženklą, atitinkamai atkuria suspausto spektro $\tilde{Y}$ elementą pagal formulę:

$$\tilde{Y}(i,j) = \pm 1{,}5 \cdot 2^n \qquad \textbf{(3.8)}$$

Antrasis uždavinys atliekamas tobulinimo vykdymo metu. Kai dekoderis gauna elemento $Y(i,j)$ n - tąjį reikšmingiausią bitą, suspausto spektro elementas $|\tilde{Y}(i,j)|$ yra padidinamas arba sumažinamas skaičiumi 2^{n-1} priklausomai nuo bito reikšmės („1" padidina, „0" sumažina).

4. PROGRAMINĖS REALIZACIJOS YPATUMAI

Programos realizacijai pasirinkome C++ programavimo kalbą. Paleidus programą, atsidaro toks langas:



4.1 pav. Programos langas

Spaudžiame „Failas -> Atidaryti nuotrauką“ ir pasirenkame norimą vaizdą. Pasirenkama nuotrauka privalo būti *.bmp formato, 8 bitų su pilka šviesos intensyvumo skale. Taip pat, vaizdo aukštis N_1 ir plotis N_2 pikseliais turi tenkinti sąlygas: $N_1 = 2^{n_1}$ ir $N_2 = 2^{n_2}$. Kai nuotrauka pasirinkta, programos lange ji rodoma maksimalia 256x256 (pikseliais) rezoliucija (vietos taupymo sumetimais). Vaizdas nėra iškraipomas, kadangi išlaikoma nuotraukos matmenų proporcija.

Sekantis žingsnis, kurį reikia atlikti, yra nuotraukos spalvų kodų surašymas į matricą. Tai galime padaryti pasirinkę meniu punktą „Rasti -> Vaizdą”. Po šios operacijos, meniu punkto „Rasti -> Spektrą” pagalba, randame nuotraukos spektrą. Spektro ryškumą keičiame slinkties juostos

4.2 pav. Slinkties juosta

pagalba. Esant poreikiui, mygtuko  pagalba, spektrą galime išsaugoti *.bmp formatu. Taip pat, galime išsaugoti spektro matricą *.csv formatu. *.csv formatas yra patogus tuo, kad iš šio formato bylos duomenis lengvai galime importuoti į skaičiuokles, tokias kaip Microsoft Office Excel, OpenOffice Calc ir kitos.

Sekančio žingsnio metu turime išsirinkti vaizdo fragmentą, kurio spektrą norime rasti. Tam reikia įvesti koeficientus 

(apie vaizdo fragmento išrinkimą žr. 2.1

skyrių). Įvedę šiuos koeficientus, pasirenkame meniu punktą „Rasti -> Fragmentą iš vaizdo“ arba „Rasti -> Fragmentą iš spektro“. Gauname pasirinkto vaizdo fragmento spektrą ir jo matricą, kuriuos, esant poreikiui, galime išsaugoti analogiškai kaip išsaugojome spektrą ir jo matricą. Taip pat, atlikus vieną ar kitą iš šių operacijų, programos lango apačioje parodoma atitinkamo algoritmo atlikimo trukmė sekundėmis.

Meniu punkte „Apie“ rasime informaciją apie programą: pagrindinė darbo užduotis, pastabos, autorius ir panašiai.

5. EKSPERIMENTAS IR JO REZULTATŲ ANALIZĖ

Šiame skyrelyje atliksime tris eksperimento dalis:

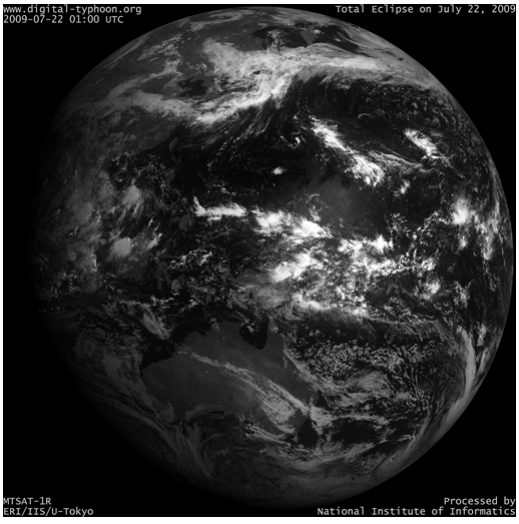
1. Palyginsime vaizdo fragmentų radimo trukmes dviem atvejais:

a) kai vaizdo fragmentas randamas iš pradinio vaizdo, naudojant nemodifikuotą Haar transformaciją, pritaikytą darbui su realiais skaičiais.

b) kai vaizdo fragmentas randamas iš viso vaizdo spektro, remiantis viena iš Haar spektro savybių – lokalizavimu erdvėje (su reliais skaičiais).

2. Pasinaudoję SPIHT metodu, ištirsime vaizdo suspaudimo efektyvumą diskrečiosios Haar transformacijos spektrinėje srityje.

Paimkime dvimatį skaitmeninį vaizdą su pilka šviesos intensyvumo skale $[X(m_1, m_2)]$, pavyzdžiui „earth.bmp“:



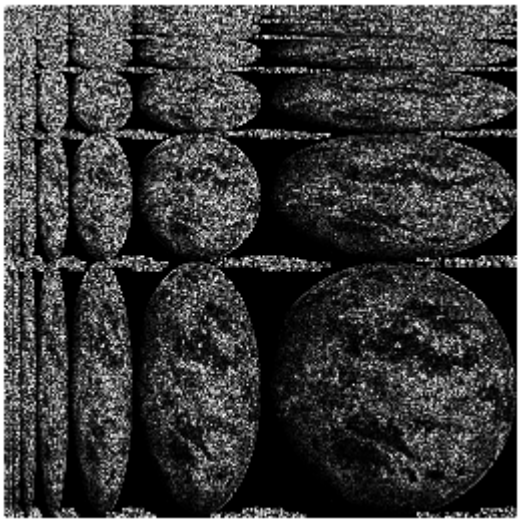
5.1 pav. Vaizdo $[X(m_1, m_2)]$ grafinis vaizdavimas, kai $N_1 = N_2 = 1024$

5.1 lentelė

Vaizdo $[X(m_1, m_2)]$ matrica, kai $N_1 = N_2 = 1024$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...	1023
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
5	19	46	16	0	0	0	0	0	0	0	6	40	34	43	16	...	0
6	66	232	105	0	0	13	45	23	0	0	49	210	125	221	106	...	0
7	34	222	146	0	0	98	238	148	0	0	84	213	60	215	146	...	0
8	8	193	178	1	0	153	226	189	10	0	123	192	13	190	178	...	0
9	0	161	207	14	12	192	131	210	45	0	165	158	0	159	207	...	0
10	0	121	227	37	53	202	42	208	88	0	196	113	0	120	227	...	0
11	0	83	235	63	103	176	2	179	139	22	213	73	0	83	235	...	0
12	0	51	227	102	157	124	0	130	180	66	212	36	0	51	227	...	0
13	0	20	205	164	195	68	0	79	204	128	190	7	0	20	205	...	0
14	0	2	173	223	198	20	0	35	204	208	152	0	0	2	173	...	0
15	0	0	145	251	164	0	0	4	186	251	113	0	0	0	145	...	0
16	0	0	70	154	73	0	0	0	93	152	50	0	0	0	70	...	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
21	0	0	0	0	0	4	4	0	0	0	0	0	0	0	0	...	0
22	0	0	28	108	166	192	191	150	60	0	0	0	0	0	0	...	0
23	0	0	121	238	178	138	155	222	227	95	0	0	0	0	0	...	0
24	0	0	58	53	1	0	0	64	217	210	31	0	0	0	51	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1023	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0

Su šiuo vaizdu atliekame diskrečiąją Haar transformaciją, kurios rezultatas yra spektras $[Y(k_1, k_2)]$:



5.2 pav. Spektro $[Y(k_1, k_2)]$ grafinis vaizdavimas, kai $N_1 = N_2 = 1024$

5.2 lentelė

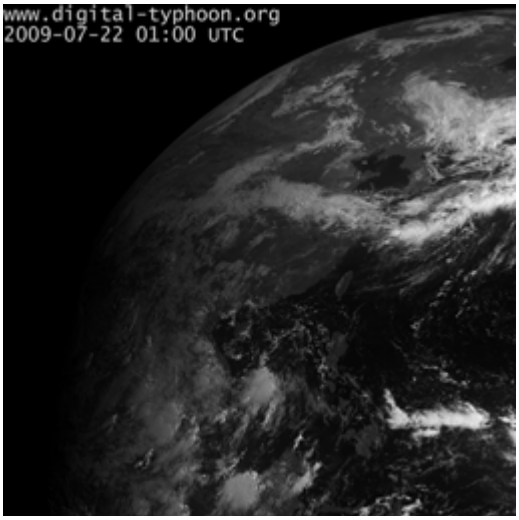
Spektro $[Y(k_1, k_2)]$ matrica, kai $N_1 = N_2 = 1024$

	0	1	2	3	4	5	6	7	8	9	...	1023
0	39.37	-11.97	-8.35	12.09	-4.81	-3.99	1.34	4.01	-0.84	-1.23	...	0.01
1	6.72	2.72	-2.40	1.24	-2.47	-1.19	0.17	0.21	-0.33	-0.62	...	0.00
2	-1.09	5.69	-6.01	3.51	2.01	-2.83	3.25	2.00	0.79	-0.54	...	0.01
3	7.52	-3.13	-0.71	0.75	-1.95	-0.31	-0.92	0.26	-0.64	-0.30	...	-0.01
4	-1.47	-1.93	2.05	2.94	1.76	-0.19	-1.04	-1.87	0.19	0.65	...	0.01
5	-6.19	1.90	0.85	-2.26	0.97	1.40	-5.43	3.95	0.33	-0.27	...	0.00
6	4.16	-3.70	0.87	4.53	-0.02	0.45	0.40	2.47	-0.55	0.69	...	0.00
7	3.29	-1.32	-2.20	1.27	-0.83	0.01	1.23	1.82	-0.19	-0.61	...	-0.01
8	-4.87	2.06	4.37	-4.55	0.67	0.85	0.87	-1.29	0.11	0.88	...	0.01
9	2.43	-2.93	-0.12	2.57	0.13	-0.53	1.62	1.80	0.14	-0.31	...	0.00
10	0.17	-0.75	-0.38	-0.46	0.31	-0.46	-0.31	1.66	0.17	-0.23	...	0.00
11	-0.69	2.27	-1.35	-2.89	0.06	-2.04	2.39	-1.92	0.04	0.53	...	0.00
12	2.66	-1.67	0.41	1.41	-0.46	-0.52	0.42	0.92	-0.19	0.70	...	0.00
13	-1.12	0.11	0.86	-2.10	0.03	0.88	0.21	-0.07	-0.09	0.44	...	0.00
14	2.45	-1.39	-0.29	0.93	-0.32	-0.29	0.40	-0.44	0.00	-0.32	...	0.00
15	0.46	-0.67	-1.62	2.43	-0.73	-0.37	0.26	1.18	-0.28	-0.36	...	-0.01
16	0.88	0.11	2.13	-1.41	0.21	-0.29	0.34	-0.08	0.07	0.08	...	0.02
17	-4.30	2.02	1.04	-2.87	0.40	1.63	-0.87	-1.13	0.00	0.53	...	0.00
18	2.10	-1.96	-0.76	1.83	0.46	-0.46	-0.85	0.78	0.01	0.20	...	0.00
19	1.42	2.43	-2.23	-0.53	-0.42	-0.10	0.14	0.24	0.05	-0.26	...	0.00
20	0.32	0.38	-0.59	0.17	-0.02	-0.07	0.91	0.16	0.06	-0.07	...	0.00
21	0.20	-1.34	0.06	0.93	0.46	-0.02	-1.74	0.53	0.04	0.31	...	0.00
22	-2.39	3.21	-0.15	-1.06	-0.39	0.30	-3.25	0.30	0.06	-0.65	...	0.00
23	0.41	-0.55	0.02	-0.26	0.18	-0.09	2.23	-0.88	-0.11	0.05	...	0.00
24	1.81	-1.70	0.47	1.55	-0.33	0.62	0.23	1.06	-0.02	0.14	...	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...
1023	0.02	-0.02	0.02	0.00	-0.02	0.00	0.00	0.00	0.00	0.04	...	0.00

čia matricos elementų reikšmės pateikiamos tikslumu 10^{-2}.

Rasime vaizdo fragmento spektrą $\left[Y^{(k_1,k_2)}(u,v)\right]$ dviem skirtingais metodais, palyginsime gautus rezultatus ir skaičiavimo procedūrų greičius.

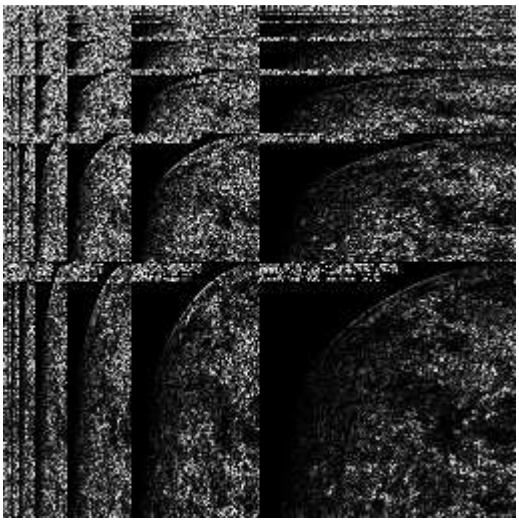
1. Iš vaizdo $[X(m_1,m_2)]$ parenkame fragmentą $\left[X^{(k_1,k_2)}(m_1,m_2)\right]$ ir randame šio fragmento spektrą $\left[Y^{(k_1,k_2)}(u,v)\right]$, pasinaudodami diskrečiąją Haar transformaciją. Tarkime, kad parinktas vaizdo fragmentas yra toks:



5.3 pav. Vaizdo fragmento $\left[X^{(2,2)}(m_1,m_2)\right]$ grafinis vaizdavimas

čia vaizdo fragmento $\left[X^{(2,2)}(m_1,m_2)\right]$ matricos elementai yra lygūs atitinkamai vaizdo $[X(m_1,m_2)]$ matricos elementams.

Parinktam vaizdo fragmentui $\left[X^{(k_1,k_2)}(m_1,m_2)\right]$ atliekame diskrečiąją Haar transformaciją (žr. 1.3 skyrių), kurios rezultatas yra šio fragmento spektras $\left[Y^{(k_1,k_2)}(u,v)\right]$:



5.4 pav. Vaizdo fragmento $[X^{(2,2)}(m_1, m_2)]$ spektro $[Y^{(2,2)}(u, v)]$ grafinis vaizdavimas, gautas 1 būdu

5.3 lentelė

Vaizdo fragmento $[X^{(2,2)}(m_1, m_2)]$ spektro $[Y^{(2,2)}(u, v)]$ matrica, gauta 1 būdu

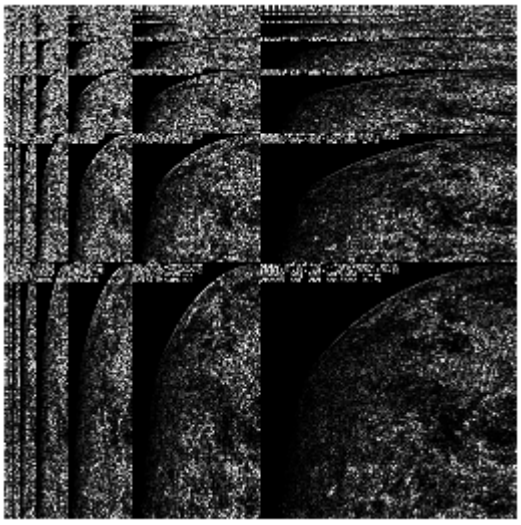
	0	1	2	3	4	5	6	7	8	9	...	511
0	36.83	-15.20	-10.29	-7.34	-1.66	-2.63	0.91	-1.90	0.12	-1.17	...	-0.02
1	6.51	-12.02	4.02	-5.67	1.58	-1.07	-3.94	-2.49	0.27	0.77	...	-0.05
2	-4.80	4.10	3.52	-0.38	0.38	1.30	-0.34	1.07	0.27	0.28	...	0.04
3	-6.06	1.71	1.95	2.80	0.67	-0.55	-0.46	0.13	0.09	0.22	...	0.00
4	-3.97	8.74	1.33	1.71	0.23	1.77	-0.06	0.62	0.38	0.00	...	-0.02
5	-0.70	-0.24	0.26	-1.07	0.29	-0.63	1.71	-5.63	0.00	0.38	...	-0.07
6	-0.83	-0.77	0.63	-0.92	0.34	-0.46	-0.68	-0.14	0.01	0.11	...	0.02
7	2.24	-2.71	0.12	-4.08	0.08	1.06	-0.90	-0.16	0.05	-0.15	...	0.01
8	1.41	4.27	0.42	-0.57	0.13	0.16	1.78	-1.49	0.19	0.21	...	0.01
9	-3.21	2.08	0.80	3.25	0.00	1.07	0.39	3.60	0.00	0.00	...	0.01
10	0.20	-1.52	0.92	-0.92	0.01	0.39	-0.99	-0.10	0.00	0.02	...	0.07
11	5.45	-4.46	-0.84	-0.20	0.10	-0.52	-1.22	-2.53	0.00	0.10	...	-0.01
12	1.00	-1.18	-0.05	-0.14	0.13	-0.13	0.17	0.12	0.00	0.06	...	0.03
13	-1.62	0.13	0.93	-0.04	0.08	0.62	-0.73	-0.06	0.01	-0.05	...	-0.02
14	1.15	-0.31	-0.79	0.60	0.13	-1.30	0.81	0.42	0.01	0.06	...	0.04
15	-0.20	0.04	0.35	-0.17	-0.22	0.10	0.77	-0.93	0.03	-0.20	...	-0.01
16	-0.46	2.00	-0.04	1.87	-0.27	-0.66	0.73	-0.06	0.54	-0.30	...	0.00
17	1.12	1.03	0.05	-0.87	0.14	0.31	-0.62	0.54	0.25	-0.14	...	-0.02
18	-1.73	1.22	0.36	1.08	0.00	0.51	-0.36	-0.37	0.00	0.00	...	0.00
19	-0.44	0.22	0.15	0.04	0.00	0.12	0.78	0.35	0.00	0.00	...	-0.01
20	1.00	-1.43	0.30	-0.97	0.00	0.02	-0.94	0.42	0.00	0.00	...	-0.01
21	-1.72	1.61	0.05	0.27	0.02	-0.15	-0.97	1.57	0.00	0.02	...	-0.06
22	2.32	-1.87	-0.39	-0.74	0.05	-0.43	0.91	-3.71	0.00	0.07	...	0.00
23	2.01	-1.80	-0.15	-1.37	0.00	-0.35	-0.69	2.47	0.00	-0.02	...	0.02
24	0.49	-0.46	-0.02	-0.30	0.00	-0.09	0.22	-0.46	0.00	-0.03	...	0.05
...	...	...	...	...	...	...	...	...	...	...	...	...
511	0.05	-0.01	-0.03	0.03	0.00	0.01	0.02	-0.08	0.00	0.00	...	-0.01

čia matricos elementų reikšmės pateikiamos tikslumu 10^{-2}.

Šiai procedūrai atlikti prireikė 0,249 sekundės.

Randame vaizdo fragmento $[X^{(k_1,k_2)}(m_1, m_2)]$ spektrą $[Y^{(k_1,k_2)}(u, v)]$ iš viso vaizdo spektro, remdamiesi spektrinio koeficiento $[Y(k_1, k_2)]$ savybėmis (žr. 1.4 skyrių), tai yra, realizuojame fragmentinio HT spektro radimo algoritmą (modifikuotą versiją) (žr. 2.3 skyrių).

Akivaizdu, kad šio algoritmo realizacijos rezultatas yra vaizdo fragmento spektras $\left[Y^{(k_1,k_2)}(u,v)\right]$:



5.5 pav. Vaizdo fragmento $\left[X^{(2,2)}(m_1,m_2)\right]$ spektro $\left[Y^{(2,2)}(u,v)\right]$ grafinis vaizdavimas, gautas 2 būdu

5.4 lentelė

Vaizdo fragmento $\left[X^{(2,2)}(m_1,m_2)\right]$ spektro $\left[Y^{(2,2)}(u,v)\right]$ matrica, gauta 2 būdu

	0	1	2	3	4	5	6	7	8	9	10	...	511
0	36.83	-15.20	-10.29	-7.34	-1.66	-2.63	0.91	-1.90	0.12	-1.17	-2.10	...	-0.02
1	6.51	-12.02	4.02	-5.67	1.58	-1.07	-3.94	-2.49	0.27	0.77	0.30	...	-0.05
2	-4.80	4.10	3.52	-0.38	0.38	1.30	-0.34	1.07	0.27	0.28	0.39	...	0.04
3	-6.06	1.71	1.95	2.80	0.67	-0.55	-0.46	0.13	0.09	0.22	0.60	...	0.00
4	-3.97	8.74	1.33	1.71	0.23	1.77	-0.06	0.62	0.38	0.00	-0.55	...	-0.02
5	-0.70	-0.24	0.26	-1.07	0.29	-0.63	1.71	-5.63	0.00	0.38	-0.19	...	-0.07
6	-0.83	-0.77	0.63	-0.92	0.34	-0.46	-0.68	-0.14	0.01	0.11	-0.17	...	0.02
7	2.24	-2.71	0.12	-4.08	0.08	1.06	-0.90	-0.16	0.05	-0.15	0.15	...	0.01
8	1.41	4.27	0.42	-0.57	0.13	0.16	1.78	-1.49	0.19	0.21	-0.70	...	0.01
9	-3.21	2.08	0.80	3.25	0.00	1.07	0.39	3.60	0.00	0.00	0.05	...	0.01
10	0.20	-1.52	0.92	-0.92	0.01	0.39	-0.99	-0.10	0.00	0.02	-0.26	...	0.07
11	5.45	-4.46	-0.84	-0.20	0.10	-0.52	-1.22	-2.53	0.00	0.10	0.06	...	-0.01
12	1.00	-1.18	-0.05	-0.14	0.13	-0.13	0.17	0.12	0.00	0.06	-0.27	...	0.03
13	-1.62	0.13	0.93	-0.04	0.08	0.62	-0.73	-0.06	0.01	-0.05	-0.05	...	-0.02
14	1.15	-0.31	-0.79	0.60	0.13	-1.30	0.81	0.42	0.01	0.06	0.32	...	0.04
15	-0.20	0.04	0.35	-0.17	-0.22	0.10	0.77	-0.93	0.03	-0.20	-0.27	...	-0.01
16	-0.46	2.00	-0.04	1.87	-0.27	-0.66	0.73	-0.06	0.54	-0.30	-0.32	...	0.00
17	1.12	1.03	0.05	-0.87	0.14	0.31	-0.62	0.54	0.25	-0.14	-0.10	...	-0.02
18	-1.73	1.22	0.36	1.08	0.00	0.51	-0.36	-0.37	0.00	0.00	0.00	...	0.00
19	-0.44	0.22	0.15	0.04	0.00	0.12	0.78	0.35	0.00	0.00	0.07	...	-0.01

20	1.00	-1.43	0.30	-0.97	0.00	0.02	-0.94	0.42	0.00	0.00	0.13	...	-0.01
21	-1.72	1.61	0.05	0.27	0.02	-0.15	-0.97	1.57	0.00	0.02	-0.22	...	-0.06
22	2.32	-1.87	-0.39	-0.74	0.05	-0.43	0.91	-3.71	0.00	0.07	0.05	...	0.00
23	2.01	-1.80	-0.15	-1.37	0.00	-0.35	-0.69	2.47	0.00	-0.02	0.09	...	0.02
24	0.49	-0.46	-0.02	-0.30	0.00	-0.09	0.22	-0.46	0.00	-0.03	-0.03	...	0.05
...	...	...	...	...	...	...	...	...	...	...	...	...	...
511	0.05	-0.01	-0.03	0.03	0.00	0.01	0.02	-0.08	0.00	0.00	0.03	...	-0.01

čia matricos elementų reikšmės pateikiamos tikslumu 10^{-2}.

Šiai procedūrai atlikti prireikė 0,093 sekundės.

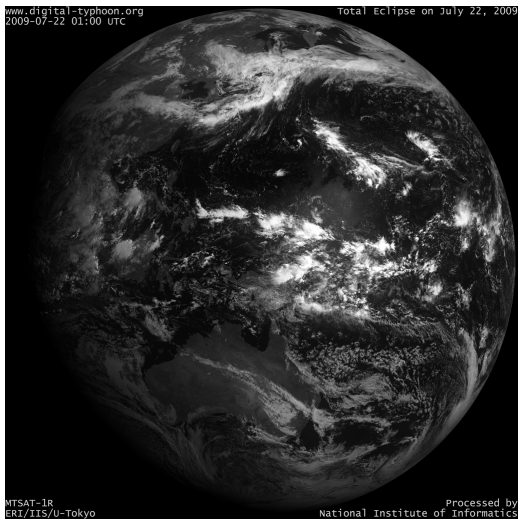
Pastebėkime, kad tiek pirmuoju, tiek antruoju metodais gauti rezultatai yra identiški, tačiau antrasis metodas yra efektyvesnis už pirmąjį, kadangi jam atlikti prireikė mažiau laiko.

Tikslesniam greičio nustatymui paimkime dar 4 papildomas nuotraukas:

5.6 pav. „poker.bmp“, kai $N_1 = N_2 = 1024$

5.7 pav. „suzuki.bmp“, kai $N_1 = N_2 = 1024$

5.8 pav. „rainbow.bmp“, kai $N_1 = N_2 = 1024$



5.9 pav. „earth.bmp“, kai $N_1 = N_2 = 1024$

Su kiekviena iš šių nuotraukų atliksime aukščiau šiame skyriuje aprašytas procedūras ir gautus rezultatus pateiksime lentelėje:

5.5 lentelė

Dviem metodais gauto vaizdo fragmento radimo greičių palyginimas

Fragmento dydis, px	3.1 pav.		3.6 pav.		3.7 pav.		3.8 pav.		3.9 pav.	
	1 met. laikas, msec	2 met. laikas, msec	1 met. laikas, msec	2 met. laikas, msec	1 met. laikas, msec	2 met. laikas, msec	1 met. laikas, msec	2 met. laikas, msec	1 met. laikas, msec	2 met. laikas, msec
512x512	253	94	248	94	256	94	249	94	228	59
256x256	158	21	154	20	153	21	156	23	108	9
128x128	110	8	100	7	112	6	114	7	43	7
64x64	97	3	95	2	100	2	98	3	31	2

2. Tirsime SPIHT algoritmo efektyvumą. Tam paimsime 256 x 256 matmenų vaizdą $[X(m_1, m_2)]$ „lena.bmp“

5.10 pav. „lena.bmp“, VKP = 0

Čia VKP yra vidutinė kvadratinė paklaida, lyginant vaizdą su originaliu vaizdu. Kuo ši paklaida didesnė, tuo vaizdas labiau suspaustas.

5.6 lentelė

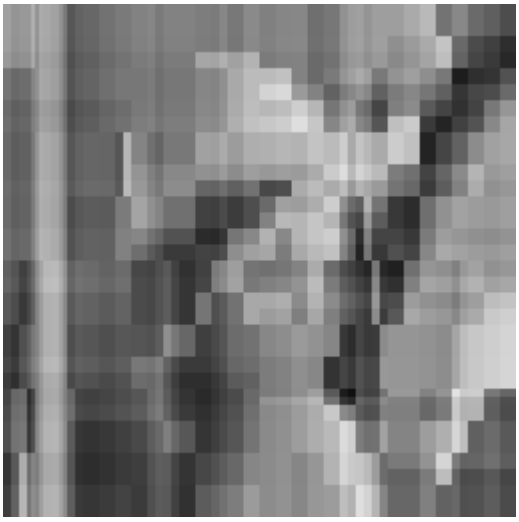
Vaizdo „lena.bmp“ 256x256 SPIHT efektyvumo tyrimas

Eilės nr.	Atkurtų bitų skaičius vienam pikseliui (max. 23)	Užkodavimo laikas, s	Dekodavimo laikas, s	k_1	k_2	VKP	Pastabos
1	**10**	101	4,019	1	1	567,09	
2	**12**	101	4,366	1	1	316,76	
3	**13**	101	4,368	1	1	222,06	
4	**15**	101	5,089	1	1	89,67	
5	**20**	101	7,754	1	1	1,42	
6	**23**	101	100	1	1	0	Originalus vaizdas
7	**13**	101	4,39	2	1	110,35	Kairė pusė vaizdo geros kokybės
8	**13**	101	4,377	2	3	159,54	Viršutinis dešinys vaizdo ketvirtis geros kokybės
9	**13**	101	4,377	3	3	174,33	Apatinis dešinys vaizdo ketvirtis geros kokybės

Iš lentelėje pateiktų duomenų matyti, kad užkodavimo laikas visada toks pats. Taip yra todėl, kad vaizdą visada užkoduojame pilnai, siekdami išvengti informacijos praradimo jį

atkuriant. ir parodo, kuris vaizdo fragmentas bus atkurtas pilnai, tai yra, išliks originalaus vaizdo kokybės (žr. 2.1 skyrelį).

Pateikiame SPIHT algoritmo dekodavimo metu gautus suspaustus vaizdus:



5.11 pav. 1. „lena.bmp“, VKP = 567,09

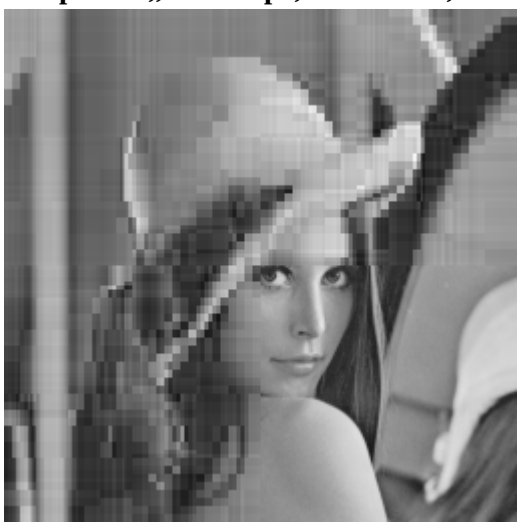
5.12 pav. 2. „lena.bmp“, VKP = 316,76

5.13 pav. 3. „lena.bmp“, VKP = 222,06

5.14 pav. 4. „lena.bmp“, VKP = 89,67

5.15 pav. 5. „lena.bmp“, VKP = 1,42

5.16 pav. 7. „lena.bmp“, VKP = 222,06



5.17 pav. 8. „lena.bmp“, VKP = 1,42

5.18 pav. 9. „lena.bmp“, VKP = 174,33

6. IŠVADOS

1. Diskrečioji Haar bangelių transformacija turi labai svarbią savybę praktiniu požiūriu – skaitmeniniame signale (vaizde) sukaupta informacija išdėstoma ir pagal dažnį, ir erdvėje.

2. Modifikuotas rekurentinis fragmentinio Haar spektro radimo algoritmas supaprastino programos realizaciją C++ aplinkoje.

3. Vaizdo fragmento spektras iš mažesnių matmenų vaizdo tiek vienu, tiek ir kitu metodais randamas per daug trumpesnį laiką, nei analogiškas vaizdo fragmentas iš didesnių matmenų vaizdo.

4. Skirtingiems vienodų matmenų vaizdams, jų fragmentų spektrų radimui, naudojant tiek vieną, tiek ir kitą metodus, procedūrų atlikimo greičiai niekaip nepriklauso nuo vaizdo specifikos (elementų reikšmių).

5. Ieškant vaizdo fragmento spektro iš viso to vaizdo spektro (žr. 3 skyrių, 2 metodas), skaičiavimai atliekami žymiai greičiau, negu ieškant to paties vaizdo fragmento spektro, naudojant diskrečiąją Haar transformaciją (žr. 3 skyrių, 1 metodas).

6. Haar transformacijos greito skaičiavimo algoritmas Andrews modifikuotas taip, kad atlikus Haar transformaciją, spektro elementai yra tik sveikieji skaičiai. Modifikavimo privalumas toks, kad Haar transformacija veikia su SPIHT algoritmu. Trūkumas – iš tokio spektro negalime išrinkti norimo vaizdo fragmento, kadangi prarasta Haar spektro lokalizavimo erdvėje savybė.

7. Haar transformacija ir SPIHT algoritmas teoriškai nėra suderinami tarpusavyje, kadangi SPIHT algoritmas dirba tik su sveikaisiais skaičiais, kai tuo tarpu Haar transformacijos metu gauti rezultatai yra realieji skaičiai. Nepaisant to, praktiškai šių dviejų algoritmų veikimą iš dalies pavyko suderinti, tačiau jis yra neefektyvus, nes vaizdo užkodavimas trunka ilgai ir programa veikia su vaizdais iki matmenų 256 x 256.

LITERATŪRA

1. Jonas Valantinas. Diskrečiosios transformacijos (mokomoji knyga), Technologija, 2009.

2. James S. Walker. A Primer on Wavelets and their Scientific Applications, Second Edition,

Chapman and Hall/CRC, 1999, p. 5-33.

3. http://www.cis.udel.edu/~amer/CISC651/IEEEwavelet.pdf

4. http://cas.ensmp.fr/~chaplais/Wavetour_presentation/transformees/Transforms.html

5. http://www.math.okstate.edu/~wrightd/5593/waveletsI/node1.html

6. http://www.dtic.upf.edu/~xserra/cursos/TDP/referencies/Park-DWT.pdf

7. http://cam.mathlab.stthomas.edu/wavelets/pdffiles/USF09/DigitalImages.pdf

8. ttp://cam.mathlab.stthomas.edu/wavelets/pdffiles/USF09/HaarWaveletTransform.pdf

10. http://www.cipr.rpi.edu/research/SPIHT/EW_Code/csvt96_sp.pdf

11. http://www.cipr.rpi.edu/~pearlman/papers/ex_spiht-ezw.pdf

12. http://www.cipr.rpi.edu/~pearlman/papers/Part1_SPC.pdf

13. http://www.cipr.rpi.edu/~pearlman/

1 PRIEDAS. PROGRAMA C++ KALBA

```
//---------------------------------------------------------------------------
--

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include <math.h>


//---------------------------------------------------------------------------
--
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
--
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
--
void   TForm1::AllocMemForDoubleMatrix(double   **&A,   int   RowCount,   int
ColCount)
// Allocates memory for new dynamic matrix
{
  A = new double*[RowCount];
  for (int i = 0; i < RowCount; i++)
    A[i] = new double[ColCount];

  for (int i = 0; i < RowCount; i++)
    for (int j = 0; j < ColCount; j++)
      A[i][j] = 0;
}
//---------------------------------------------------------------------------
--
//(120515+++ actions with double)
void TForm1::FindSpectrumDouble(double **X, double **Y, int m, int n, bool
Rows)
{
  double* T = new double[n];

  int itercount = Log2(n);  // Iteration count

  for (int i = 0; i < m; i++)
  {
    int tmp_n = n;
    // Copy matrix collumn to vector
    for (int j = 0; j < n; j++)
      if (Rows)
        T[j] = Y[i][j];
      else
        T[j] = X[j][i];
    for (int s = 0; s < itercount; s++)
    {
      int k = 0;
      for (int j = 0; j < tmp_n; j += 2)
      {
        if (Rows)
```

```
        {
          Y[i][k] = T[j] + T[j+1];
          Y[i][k+(tmp_n/2)] = T[j] - T[j+1];
        }
        else
        {
          Y[k][i] = T[j] + T[j+1];
          Y[k+(tmp_n/2)][i] = T[j] - T[j+1];
        }
        k++;
      }

      tmp_n = (int)tmp_n / 2;

      for (int j = 0; j < tmp_n; j++)
        if (Rows)
          T[j] = Y[i][j];
        else
          T[j] = Y[j][i];

      // Multiply by weight
      if (tmp_n != 1)
      {
        for (int j = tmp_n; j < n; j++)
          if (Rows)
            Y[i][j] = Y[i][j]*sqrt(2);
          else
            Y[j][i] = Y[j][i]*sqrt(2);
      }
    }
  }
  delete []T;

  // Multiply spectrum elements by 1/n
      for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
          if (Rows)
            Y[i][j] = Y[i][j] / n;
          else
            Y[j][i] = Y[j][i] / n;
}
//---------------------------------------------------------------------------
--
void  TForm1::FindViewDouble(double  **X,  double  **Y,  int  m,  int  n,  bool
Rows)
{
double* T = new double[n];

  int itercount = Log2(n);  // Iteration count

  for (int i = 0; i < m; i++)
  {
    int tmp_n = 2;

    // Copy matrix collumn to vector
    for (int j = 0; j < n; j++)
      if (Rows)
        T[j] = Y[i][j];
      else
        T[j] = X[j][i];

    for (int s = 0; s < itercount; s++)
    {
      // Multiply by weight
```

```
      for (int j = tmp_n; j < n; j++)
        T[j] = T[j] * sqrt(2);

      int k = 0;
      for (int j = 0; j < tmp_n; j += 2)
      {
        if (Rows)
        {
          X[i][j] = T[k] + T[k+tmp_n/2];
          X[i][j+1] = T[k] - T[k+tmp_n/2];
        }
        else
        {
          X[j][i] = T[k] + T[k+tmp_n/2];
          X[j+1][i] = T[k] - T[k+tmp_n/2];
        }
        k++;
      }

      for (int j = 0; j < tmp_n; j++)
        if (Rows)
          T[j] = X[i][j];
        else
          T[j] = X[j][i];

      tmp_n = (int)tmp_n * 2;
    }
  }
  delete []T;
}
//---------------------------------------------------------------------------
--
void   TForm1::PrintToStringGridDouble(double   **A,   int   RowCount,   int
ColCount, TStringGrid *SG)
{
  SG->RowCount = RowCount+1;
  SG->ColCount = ColCount+1;
  for (int i = 1; i <= RowCount; i++)
  {
    SG->Cells[0][i] = i-1;
    SG->Cells[i][0] = i-1;
    for (int j = 1; j <= ColCount; j++)
    {
      SG->Cells[j][i] = A[i-1][j-1];
    }
  }
}
//---------------------------------------------------------------------------
--
void  TForm1::DrawToCanvasDouble(double  **A,  int  RowCount,  int  ColCount,
TImage *IM)
{
  //Clear Image Canvas
  for (int i = 0; i < Height; i++)
    for (int j = 0; j < Width; j++)
      IM->Canvas->Pixels[j][i] = (TColor)RGB(255,255,255);

   for (int i = 0; i < RowCount; i++)
    for (int j = 0; j < ColCount; j++)
      IM->Canvas->Pixels[j][i]                                          =
(TColor)RGB(fabs(A[i][j])*pow(2,11),fabs(A[i][j])*pow(2,11),fabs(A[i][j])*p
ow(2,11));
}
```

```
//---------------------------------------------------------------------------
--
void TForm1::ChangeLuminanceDouble(double **A, int RowCount, int ColCount,
                              int multiplier, TStringGrid *SG, TImage *IM)
{
  for (int i = 0; i < RowCount; i++)
    for (int j = 0; j < ColCount; j++)
    {
      IM->Canvas->Pixels[j][i] = (TColor)RGB(255,255,255);
      if (multiplier != 1)
      {
        double absolute = fabs(A[i][j]) * multiplier;

        //if (absolute - 0.000001 < 0) {
          //absolute = 0.000001;
          //absolute = absolute * multiplier;
        //}

        if (absolute > 255)
          absolute = 255;

        if ((i+1 < RowCount) && (j+1 < ColCount))
        SG->Cells[j+1][i+1] = absolute;
        IM->Canvas->Pixels[j][i] = (TColor)RGB(absolute,absolute,absolute);
      }
      else
      {
        SG->Cells[j+1][i+1] = A[i][j];
        IM->Canvas->Pixels[j][i] = (TColor)RGB(A[i][j],A[i][j],A[i][j]);
      }
    }
}
//---------------------------------------------------------------------------
--
void  TForm1::SaveToCSVDouble(double  **A,  int  RowCount,  int  ColCount,
AnsiString FileName)
{
  ofstream fr(FileName.c_str());

  fr << ";";
  for (int j = 0; j < ColCount; j++)
    fr << j << ";";
  fr << endl;

  for (int i = 0; i < RowCount; i++)
  {
    fr << i << ";";
    for (int j = 0; j < ColCount; j++)
      fr << A[i][j] << ";";
    fr << endl;
  }
  fr.close();

  /*FR = fopen(FileName.c_str(), "w");
  for (int i = 0; i < RowCount; i++)
  {
    for (int j = 0; j < ColCount; j++)
      fprintf(FR, "%d", FloatToStr(A[i][j]), ";");
    fprintf(FR, "\n");
  }
  fclose(FR);*/
}
//---------------------------------------------------------------------------
--
```

```
void TForm1::FindPartFromViewDouble(int k1, int k2, int &lh, int &lw)
{
  bool firsttime = true;
  int StartRow, EndRow,
      StartCol, EndCol,
      RowCount, ColCount;

  for (int i = 0; i < Height; i++)
    if (H1[k1][i] != 0)
    {
      if (firsttime)
      {
        StartRow = i;
        firsttime = false;
      }
      EndRow = i;
    }

  firsttime = true;
  for (int i = 0; i < Width; i++)
    if (H2[k2][i] != 0)
    {
      if (firsttime)
      {
        StartCol = i;
        firsttime = false;
      }
      EndCol = i;
    }

  lh = EndRow - StartRow + 1;  // Fragment row count
  lw = EndCol - StartCol + 1;  // Fragnent collumn count

  AllocMemForDoubleMatrix(XfragFViewDbl,lh,lw);

  for (int i = 0; i < lh; i++)
    for (int j = 0; j < lw; j++)
      XfragFViewDbl[i][j] = XDbl[StartRow+i][StartCol+j];

  AllocMemForDoubleMatrix(YfragFViewDbl, lh, lw);

  //Find fragment spectrum
  FindSpectrumDouble(XfragFViewDbl,  YfragFViewDbl,  lw,  lh,  false);  //
Transform Collumns
  FindSpectrumDouble(XfragFViewDbl,  YfragFViewDbl,  lh,  lw,  true);   //
Transform Rows

  //(120514+++)
  StartFragmentCordX = StartRow;
  StartFragmentCordY = StartCol;
  //(---120514)
}
//(---120515)
//---------------------------------------------------------------------------
--
void TForm1::FindSpectrum(long long int **X, long long int **Y, int m, int
n, bool Rows)
{
  long long int* T = new long long int[n];

  int itercount = Log2(n);  // Iteration count

  for (int i = 0; i < m; i++)
  {
```

```
    int tmp_n = n;
    // Copy matrix collumn to vector
    for (int j = 0; j < n; j++)
      if (Rows)
        T[j] = Y[i][j];
      else
        T[j] = X[j][i];
    for (int s = 0; s < itercount; s++)
    {
      int k = 0;
      for (int j = 0; j < tmp_n; j += 2)
      {
        if (Rows)
        {
          Y[i][k] = T[j] + T[j+1];
          Y[i][k+(tmp_n/2)] = T[j] - T[j+1];
        }
        else
        {
          Y[k][i] = T[j] + T[j+1];
          Y[k+(tmp_n/2)][i] = T[j] - T[j+1];
        }
        k++;
      }

      tmp_n = (int)tmp_n / 2;

      for (int j = 0; j < tmp_n; j++)
        if (Rows)
          T[j] = Y[i][j];
        else
          T[j] = Y[j][i];

      // Multiply by weight (12.03.10+++ commented}
      /*
      if (tmp_n != 1)
      {
        for (int j = tmp_n; j < n; j++)
          if (Rows)
            Y[i][j] = Y[i][j]*sqrt(2);
          else
            Y[j][i] = Y[j][i]*sqrt(2);
      }
      */
      //(---12.03.10)
    }
  }
  delete []T;

  // Multiply spectrum elements by 1/n
  //(120510+++ commented because spectrum elements must be integer)
  /*
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      if (Rows)
        Y[i][j] = Y[i][j] / n;
      else
        Y[j][i] = Y[j][i] / n;
  */
  //(---120510)
}
//---------------------------------------------------------------------------
--
```

```
void TForm1::FindView(long long int **X, long long int **Y, int m, int n,
bool Rows)
{
long long int* T = new long long int[n];

  int itercount = Log2(n);  // Iteration count

  for (int i = 0; i < m; i++)
  {
    int tmp_n = 2;

    // Copy matrix collumn to vector
    for (int j = 0; j < n; j++)
      if (Rows)
        T[j] = Y[i][j];
      else
        T[j] = X[j][i];

    for (int s = 0; s < itercount; s++)
    {
      // Multiply by weight
      /* (120310+++ commented)
      for (int j = tmp_n; j < n; j++)
        T[j] = T[j] * sqrt(2);
      (---12.03.10) */

      int k = 0;
      for (int j = 0; j < tmp_n; j += 2)
      {
        if (Rows)
        {
          /* (120310+++) commented
          X[i][j] = T[k] + T[k+tmp_n/2];
          X[i][j+1] = T[k] - T[k+tmp_n/2];
          */
          X[i][j] = (T[k] + T[k+tmp_n/2]) / 2;
          X[i][j+1] = (T[k] - T[k+tmp_n/2]) / 2;
          //(---120310)
        }
        else
        {
          /* (120310+++) commented
          X[j][i] = T[k] + T[k+tmp_n/2];
          X[j+1][i] = T[k] - T[k+tmp_n/2];
          */
          X[j][i] = (T[k] + T[k+tmp_n/2]) / 2;
          X[j+1][i] = (T[k] - T[k+tmp_n/2]) / 2;
          //(---120310)
        }
        k++;
      }

      for (int j = 0; j < tmp_n; j++)
        if (Rows)
          T[j] = X[i][j];
        else
          T[j] = X[j][i];

      tmp_n = (int)tmp_n * 2;
    }
  }
  delete []T;

  //(120510+++ commented because spectrum elements must be integer)
```

```
  /*
  //(120310+++)
  // Multiply view elements by n
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      if (Rows)
        X[i][j] = X[i][j] * n; //commented
      else
        X[j][i] = X[j][i] * n;
  //(---12.03.10)
  */
  //(---120510 commented)
}
//---------------------------------------------------------------------------
void  TForm1::PrintToStringGrid(long  long  int  **A,  int  RowCount,  int
ColCount, TStringGrid *SG)
{
  SG->RowCount = RowCount+1;
  SG->ColCount = ColCount+1;
  for (int i = 1; i <= RowCount; i++)
  {
    SG->Cells[0][i] = i-1;
    SG->Cells[i][0] = i-1;
    for (int j = 1; j <= ColCount; j++)
    {
      SG->Cells[j][i] = A[i-1][j-1];
    }
  }
}
//---------------------------------------------------------------------------
void TForm1::DrawToCanvas(long long int **A, int RowCount, int ColCount,
TImage *IM)
{
  //Clear Image Canvas
  for (int i = 0; i < Height; i++)
    for (int j = 0; j < Width; j++)
      IM->Canvas->Pixels[j][i] = (TColor)RGB(255,255,255);

   for (int i = 0; i < RowCount; i++)
    for (int j = 0; j < ColCount; j++)
      //IM->Canvas->Pixels[j][i]                                          =
(TColor)RGB(fabs(A[i][j])*pow(2,11),fabs(A[i][j])*pow(2,11),fabs(A[i][j])*p
ow(2,11));
      IM->Canvas->Pixels[j][i]                                          =
(TColor)RGB(abs(A[i][j]),abs(A[i][j]),abs(A[i][j]));
}
//---------------------------------------------------------------------------
void TForm1::ChangeLuminance(long long int **A, int RowCount, int ColCount,
                             int multiplier, TStringGrid *SG, TImage *IM)
{
  for (int i = 0; i < RowCount; i++)
    for (int j = 0; j < ColCount; j++)
    {
      IM->Canvas->Pixels[j][i] = (TColor)RGB(255,255,255);
      if (multiplier != 1)
      {
        long long int absolute = abs(A[i][j]) * multiplier;

        //if (absolute - 0.000001 < 0) {
          //absolute = 0.000001;
          //absolute = absolute * multiplier;
```

```
        //}

        if (absolute > 255)
          absolute = 255;

        if ((i+1 < RowCount) && (j+1 < ColCount))
        SG->Cells[j+1][i+1] = absolute;
        IM->Canvas->Pixels[j][i] = (TColor)RGB(absolute,absolute,absolute);
      }
      else
      {
        SG->Cells[j+1][i+1] = A[i][j];
        IM->Canvas->Pixels[j][i] = (TColor)RGB(A[i][j],A[i][j],A[i][j]);
      }
    }
}
//---------------------------------------------------------------------------
--
void  TForm1::SaveToCSV(long  long  int  **A,  int  RowCount,  int  ColCount,
AnsiString FileName)
{
  ofstream fr(FileName.c_str());

  fr << ";";
  for (int j = 0; j < ColCount; j++)
    fr << j << ";";
  fr << endl;

  for (int i = 0; i < RowCount; i++)
  {
    fr << i << ";";
    for (int j = 0; j < ColCount; j++)
      fr << A[i][j] << ";";
    fr << endl;
  }
  fr.close();

  /*FR = fopen(FileName.c_str(), "w");
  for (int i = 0; i < RowCount; i++)
  {
    for (int j = 0; j < ColCount; j++)
      fprintf(FR, "%d", FloatToStr(A[i][j]), ";");
    fprintf(FR, "\n");
  }
  fclose(FR);*/
}
//---------------------------------------------------------------------------
--
void TForm1::FindHTMatrix(double **A, int N)
//Finds HT matrix
{
  int i = 0;
  int n = Log2(N);
  DiscrHaarFunctSet(A, 0, 0, N, i);
  for (int r = 0; r < n; r++)
    for (int m = 1; m <= pow(2,r); m++) {
      i++;
      DiscrHaarFunctSet(A, r, m, N, i);
    }
}
//---------------------------------------------------------------------------
--
void TForm1::DiscrHaarFunctSet(double **A, int r, int m, int N, int RowNo)
//Discretizes Haar function set
```

```
{
  int StartInd = ((m-1) / pow(2,r)) * N;
  int MiddleInd = ((m - 0.5) / pow(2,r)) * N;
  int EndInd = (m / pow(2,r)) * N;
  double elem = pow(2,0.5*r);
  for (int j = 0; j < N; j++) {
    A[RowNo][j] = 0;

    //Exception for the first line
    if (RowNo == 0)
      A[RowNo][j] = 1;

    if ((StartInd <= j) && (j < MiddleInd))
      A[RowNo][j] = elem;
    if ((MiddleInd <= j) && (j < EndInd))
      A[RowNo][j] = -elem;
  }
}

//---------------------------------------------------------------------------
--
void TForm1::FindPartFromView(int k1, int k2, int &lh, int &lw)
{
  bool firsttime = true;
  int StartRow, EndRow,
      StartCol, EndCol,
      RowCount, ColCount;

  for (int i = 0; i < Height; i++)
    if (H1[k1][i] != 0)
    {
      if (firsttime)
      {
        StartRow = i;
        firsttime = false;
      }
      EndRow = i;
    }

  firsttime = true;
  for (int i = 0; i < Width; i++)
    if (H2[k2][i] != 0)
    {
      if (firsttime)
      {
        StartCol = i;
        firsttime = false;
      }
      EndCol = i;
    }

  lh = EndRow - StartRow + 1;  // Fragment row count
  lw = EndCol - StartCol + 1;  // Fragnent collumn count

  AllocMemForLongIntMatrix(XfragFView,lh,lw);

  for (int i = 0; i < lh; i++)
    for (int j = 0; j < lw; j++)
      XfragFView[i][j] = X[StartRow+i][StartCol+j];

  AllocMemForLongIntMatrix(YfragFView, lh, lw);

  //Find fragment spectrum
```

```
  FindSpectrum(XfragFView,  YfragFView,  lw,  lh,  false);  //  Transform
Collumns
  FindSpectrum(XfragFView, YfragFView, lh, lw, true);  // Transform Rows

  //(120514+++)
  StartFragmentCordX = StartRow;
  StartFragmentCordY = StartCol;
  //(---120514)
}
//---------------------------------------------------------------------------
--
void TForm1::FillAlphaBeta(int *&A ,int k, int n)
{
  // Fills Alpha and Beta array
  A = new int[n+1];
  A[0] = k;
  for (int i = 1; i <= n; i++)
    A[i] = (int)A[i-1] / 2;
}
//---------------------------------------------------------------------------
--
int TForm1::SolveFunctionA(int k1, int k2, int s2, int n2)
{
  double a = Y[k1][0];
  if (k2 > 1)
    for (int r = 1; r <= n2 - s2; r++)
      a += pow(-1, Beta[r-1]) * sqrt(pow(2,n2-s2-r)) * Y[k1][Beta[r]];
  return(a);
}
//---------------------------------------------------------------------------
--
int TForm1::SolveFunctionB(int k1, int k2, int s1, int n1)
{
  double b = Y[0][k2];
  if (k1 > 1)
    for (int r = 1; r <= n1 - s1; r++)
      b += pow(-1, Alpha[r-1]) * sqrt(pow(2,n1-s1-r)) * Y[Alpha[r]][k2];
  return(b);
}
//---------------------------------------------------------------------------
--
void TForm1::FindSpectralCoeffs(int k1, int k2, int N)
{
  int s1, l1, n1;
  n1 = Log2(Height);
  s1 = n1 - (int)Log2(k1);
  l1 = k1 - pow(2,n1-s1);

  int s2, l2, n2;
  n2 = Log2(Width);
  s2 = n2 - (int)Log2(k2);
  l2 = k2 - pow(2,n2-s2);

  // Find array length
  int length1 = 0;
  for (int i = 0; i < s1; i++)
    length1 += pow(2,i);

  int length2 = 0;
  for (int i = 0; i < s2; i++)
    length2 += pow(2,i);

  // Allocate memory for new array
  int *T1 = new int[length1];
```

```
  int *T2 = new int[length2];

  // Find Tau
  int i = 0;
  for (int t = 1; t < s1; t++)
    for (int j = pow(2,t)*k1; j <= pow(2,t)*(k1+1)-1; j++)
    {
      i++;
      T1[i] = j;
    }

  i = 0;
  for (int t = 1; t < s2; t++)
    for (int j = pow(2,t)*k2; j <= pow(2,t)*(k2+1)-1; j++)
    {
      i++;
      T2[i] = j;
    }

  // Fill Alpha and Beta array
  FillAlphaBeta(Alpha,k1,n1-s1);
  FillAlphaBeta(Beta,k2,n2-s2);

  // Allocate memory for matrix
  AllocMemForLongIntMatrix(XfragFSpectr, Height, Width);

  // Find zero coefficient
  int kp1, kp2;
  kp1 = (int) k1 / 2;  // k1 / 2 integral part
  kp2 = (int) k2 / 2;  // k2 / 2 integral part
  XfragFSpectr[0][0] = 0;
  while ((kp1 > 1) && (kp2 > 1))
  {
    XfragFSpectr[0][0] += Y[kp1][kp2] + pow(-1,k1) * sqrt(pow(2,n1-s1-1)) *
                          SolveFunctionA(kp1, kp2, s2, n2) +
                          + pow(-1,k2) * sqrt(pow(2,n2-s2-1)) *
                          SolveFunctionB(kp1, kp2, s1, n1) +
                          + pow(-1,k1+k2) * sqrt(pow(2,n1+n2-s1-s2-2)) *
                          Y[kp1][kp2];
    kp1 = (int) kp1 / 2;
    kp2 = (int) kp2 / 2;
  }



  // Add element to Tau
  T1[0] = k1;
  T2[0] = k2;

  // Find (u,0) coeffs
  for (int u = 1; u <= pow(2,s1)-1; u++)
    XfragFSpectr[u][0]   =   sqrt(pow(2,n1-s1))   *   SolveFunctionA(T1[u-
1],k2,s2,n2);

  // Find (0,v) coeffs
  for (int v = 1; v <= pow(2,s2)-1; v++)
    XfragFSpectr[0][v]   =   sqrt(pow(2,n2-s2))   *   SolveFunctionB(k1,T2[v-
1],s1,n1);

  //Find (u,v) coeffs
  for (int u = 1; u <= pow(2,s1)-1; u++)
    for (int v = 1; v <= pow(2,s2)-1; v++)
      XfragFSpectr[u][v] = sqrt(pow(2,n1*n2-s1-s2)) * Y[T1[u-1]][T2[v-1]];
```

```
}
//---------------------------------------------------------------------------
--
void TForm1::FindSpectralCoeffsNewAlgorythm(int k1, int k2, int &fauk, int
&fplo)
{
  // Find spectral coefficients using new algorythm
  int i1, l1, n1;
  n1 = Log2(Height);
  i1 = n1 - (int)Log2(k1);
  //l1 = k1 - pow(2,n1-i1);

  int i2, l2, n2;
  n2 = Log2(Width);
  i2 = n2 - (int)Log2(k2);
  //l2 = k2 - pow(2,n2-i2);

  // Find array length
  int length1 = 0;
  for (int i = 0; i < i1; i++)
    length1 += pow(2,i);

  int length2 = 0;
  for (int i = 0; i < i2; i++)
    length2 += pow(2,i);

  // Allocate memory for new array
  int *T1 = new int[length1];
  int *T2 = new int[length2];

  // Find Tau
  // Add element to Tau
  T1[0] = k1;
  int i = 0;
  for (int t = 1; t < i1; t++)
    for (int j = pow(2,t)*k1; j <= pow(2,t)*(k1+1)-1; j++)
    {
      i++;
      T1[i] = j;
    }

  T2[0] = k2;
  i = 0;
  for (int t = 1; t < i2; t++)
    for (int j = pow(2,t)*k2; j <= pow(2,t)*(k2+1)-1; j++)
    {
      i++;
      T2[i] = j;
    }

  // Fill Alpha and Beta array
  FillAlphaBeta(Alpha,k1,n1-i1);
  FillAlphaBeta(Beta,k2,n2-i2);

  // Allocate memory for matrix
  fauk = pow(2,i1);
  fplo = pow(2,i2);
  AllocMemForDoubleMatrix(YfragFSpectrDbl, fauk, fplo);

  //Find coeff [0][0]
  YfragFSpectrDbl[0][0] = YDbl[0][0];

  for (int s = 1; s <= n1-i1; s++)
    YfragFSpectrDbl[0][0] += pow(-1,Alpha[s-1]) * sqrt(pow(2, n1-i1-s)) *
```

```
                              YDbl[Alpha[s]][0];

  for (int t = 1; t <= n2-i2; t++)
    YfragFSpectrDbl[0][0] += pow(-1,Beta[t-1]) * sqrt(pow(2, n2-i2-t)) *
                              YDbl[0][Beta[t]];

  for (int t = 1; t <= n2-i2; t++)
    for (int s = 1; s <= n1-i1; s++)
      YfragFSpectrDbl[0][0] += pow(-1,Alpha[s-1]+Beta[t-1]) *
                               sqrt(pow(2,n1+n2-i1-i2-s-t)) *
                               YDbl[Alpha[s]][Beta[t]];

  //Find coeffs [u][0]
  for (int u = 1; u <= pow(2,i1)-1; u++)
  {
    YfragFSpectrDbl[u][0] = YDbl[T1[u-1]][0];
    for (int t = 1; t <= n2-i2; t++)
      YfragFSpectrDbl[u][0] += pow(-1,Beta[t-1]) * sqrt(pow(2,n2-i2-t)) *
                               YDbl[T1[u-1]][Beta[t]];

    YfragFSpectrDbl[u][0] *= sqrt(pow(2, n1-i1));
  }

  //Find coeffs [o][v]
  for (int v = 1; v <= pow(2,i2)-1; v++)
  {
    YfragFSpectrDbl[0][v] = YDbl[0][T2[v-1]];
    for (int s = 1; s <= n1-i1; s++)
      YfragFSpectrDbl[0][v] += pow(-1,Alpha[s-1]) * sqrt(pow(2,n1-i1-s)) *
                              YDbl[Alpha[s]][T2[v-1]];

    YfragFSpectrDbl[0][v] *= sqrt(pow(2, n2-i2));
  }

  //Find coeffs [u][v]
  for (int u = 1; u <= pow(2,i1)-1; u++)
    for (int v = 1; v <= pow(2,i2)-1; v++)
      YfragFSpectrDbl[u][v]   =   sqrt(pow(2,n1+n2-i1-i2))   *   YDbl[T1[u-
1]][T2[v-1]];
}
//---------------------------------------------------------------------------
--
//(120421+++)
void TForm1::AllocMemForStringMatrix(string *&A, int lvl)
{
  A = new string[lvl];
}
//---------------------------------------------------------------------------
--
void TForm1::AllocMemForLongIntMatrix(long long int **&A, int RowCount, int
ColCount)
{
  A = new long long int*[RowCount];
  for (int i = 0; i < RowCount; i++)
    A[i] = new long long int[ColCount];

  for (int i = 0; i < RowCount; i++)
    for (int j = 0; j < ColCount; j++)
      A[i][j] = 0;
}
//---------------------------------------------------------------------------
--
void TForm1::PrintToStringGridListsCapt(TStringGrid *SG)
{
```

```
  //Prints captions in 0-th StringGrid row
  SG->FixedRows = 1;
  SG->RowCount = 1;
  SG->ColCount = CListDataCount;
  SG->Cells[0][0] = "X";
  SG->Cells[1][0] = "Y";
  SG->Cells[2][0] = "Tipas";
  SG->Cells[3][0] = "Vertė";
  SG->Cells[4][0] = "Lygmuo";
  SG->Cells[5][0] = "Iteracija";
}
//---------------------------------------------------------------------------
--
void TForm1::PrintToStringGridLists(long long int **Y, TListElement *first,
int iter, TStringGrid *SG)
{
  //Prints LIS, LIP and LSP to StringGrid
  TListElement *r;

  r = first;
  while (r != NULL)
  {
    SG->Cells[0][SG->RowCount] = r->Data.x;
    SG->Cells[1][SG->RowCount] = r->Data.y;
    SG->Cells[2][SG->RowCount] = r->Data.type;
    SG->Cells[3][SG->RowCount] = Y[r->Data.x][r->Data.y];
    SG->Cells[4][SG->RowCount] = r->Data.level;
    SG->Cells[5][SG->RowCount] = iter;
    SG->RowCount += 1;
    r = r->next;
  }
}
//---------------------------------------------------------------------------
--
int TForm1::FindMaxSpectrumElement(long long int **A, int m, int n)
{
  //Finds maximum spectrum element and calculates max level (n)
  int max = 0;
  int t;
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      if (A[i][j] > max)
        max = A[i][j];
  t = (int)Log2(abs(max));
  return(t);
  //return(pow(2,t));
}
//---------------------------------------------------------------------------
---
char TForm1::FindMostSignificantBit(long long int numb, int lvl)
{
  /*string bits;

  do
    bits.push_back( '0' + (numb & 1) );
  while (numb >>= 1);

  reverse(bits.begin(), bits.end());
  return(bits);*/

  string bits;
  int count = 0;

  do
```

```
  {
    bits.push_back( '0' + (numb & 1) );
    count++;
  }
  while ((numb >>= 1) || (lvl >= count));

  //reverse(bits.begin(), bits.end());
  return(bits[lvl]);
}
//---------------------------------------------------------------------------
---
void  TForm1::CreateList(TListElement  *&first,  TListElement  *&last,  TList
parList)
{
  //Create new list
  first = NULL;
  last = NULL;
  first = new TListElement;

  //Add first element to list
  first->Data = parList;
  last = first;
  last->next = NULL;
  last->prev = NULL;
}
//---------------------------------------------------------------------------
--
void TForm1::DeleteList(TListElement *&first, TListElement *&last)
{
  //Deletes existing list
  TListElement *r;

  while (first != NULL)
  {
    r = first;
    first = first->next;
    delete r;
  }
  last = NULL;
}
//---------------------------------------------------------------------------
--
void  TForm1::AddElementToList(TListElement  *&first,  TListElement  *&last,
TList parList)
{
  if (first == NULL)
  {
    CreateList(first,last,parList);
  }
  else
  {
    //Create and insert new element to list
    TListElement *r;
    r = new TListElement;
    r->Data = parList;
    last->next = r;
    r->prev = last;
    last = r;
    last->next = NULL;
  }
}
//---------------------------------------------------------------------------
--
```

```
void   TForm1::RemoveElementFromList(TListElement   *&first,   TListElement
*&last, TListElement *r)
{
  //Remove existing element from list
  if (first != NULL && r != NULL)
  {
    if (r == first)
      first = first->next;
    if (r == last)
      last = last->prev;
    if (r->prev != NULL)
      r->prev->next = r->next;
    if (r->next != NULL)
      r->next->prev = r->prev;
    delete r;
  }
}
//---------------------------------------------------------------------------
--
char TForm1::FindDSignificanceEnc(long long int **Y, int m, int n, int k,
int l, long long int thr)
{
  //Checks D set significance
  //m - Y row count
  //n - Y coll count
  //k - row tree root index
  //l - coll tree root index

  int lvl = 0;
  int n1 = Log2(m);
  int n2 = Log2(n);
  int nx = n1;
  if (n2 < n1)
    nx = n2;

  //Generate D set
  TListElement *Dfirst;
  TListElement *Dlast;
  TListElement *r;
  TList List;

  List.x = k;
  List.y = l;
  List.type = ' ';
  List.level = 0;
  CreateList(Dfirst,Dlast,List);

  r = Dfirst;

  while ((r != NULL) && (lvl < nx - 1))
  {

    k = r->Data.x;
    l = r->Data.y;

    if ((2*k+1 >= m) || (2*l+1 >= n))
      break;

    List.type = ' ';
    List.level = lvl + 1;

    List.x = 2*k;
    List.y = 2*l;
    AddElementToList(Dfirst, Dlast, List);
```

```
    List.x = 2*k;
    List.y = 2*l+1;
    AddElementToList(Dfirst, Dlast, List);

    List.x = 2*k+1;
    List.y = 2*l;
    AddElementToList(Dfirst, Dlast, List);

    List.x = 2*k+1;
    List.y = 2*l+1;
    AddElementToList(Dfirst, Dlast, List);

    if (abs(Y[2*k][2*l]) >= thr)
      return('1');
    if (abs(Y[2*k][2*l+1]) >= thr)
      return('1');
    if (abs(Y[2*k+1][2*l]) >= thr)
      return('1');
    if (abs(Y[2*k+1][2*l+1]) >= thr)
      return('1');

    if (r->next != NULL)
      lvl += r->next->Data.level - r->Data.level;

    r = r->next;
  }
  return('0');
}
//---------------------------------------------------------------------------
--
char TForm1::FindLSignificanceEnc(long long int **Y, int m, int n, int k,
int l, long long int thr, bool &exists)
{
  //Checks L set significance and existence
  //m - Y row count
  //n - Y coll count
  //k - row tree root index
  //l - coll tree root index
  //exists - if L exists for row k and coll l element

  int lvl = 0;
  int n1 = Log2(m);
  int n2 = Log2(n);
  int nx = n1;
  if (n2 < n1)
    nx = n2;

  //Generate L set
  TListElement *Lfirst;
  TListElement *Llast;
  TListElement *r;
  TList List;

  List.x = k;
  List.y = l;
  List.type = ' ';
  List.level = 0;
  CreateList(Lfirst,Llast,List);


  exists = false;
  r = Lfirst;
```

```
  while ((r != NULL) && (lvl < nx - 1))
  {
    k = r->Data.x;
    l = r->Data.y;

    if ((2*k+1 >= m) || (2*l+1 >= n))
      break;

    List.type = ' ';
    List.level = lvl + 1;

    List.x = 2*k;
    List.y = 2*l;
    AddElementToList(Lfirst, Llast, List);

    List.x = 2*k;
    List.y = 2*l+1;
    AddElementToList(Lfirst, Llast, List);

    List.x = 2*k+1;
    List.y = 2*l;
    AddElementToList(Lfirst, Llast, List);

    List.x = 2*k+1;
    List.y = 2*l+1;
    AddElementToList(Lfirst, Llast, List);

    //Skip O set, because L = D - O
    if (lvl != 0)
    {
      exists = true;

      if (abs(Y[2*k][2*l]) >= thr)
        return('1');
      if (abs(Y[2*k][2*l+1]) >= thr)
        return('1');
      if (abs(Y[2*k+1][2*l]) >= thr)
        return('1');
      if (abs(Y[2*k+1][2*l+1]) >= thr)
        return('1');
    }
    if (r->next != NULL)
      lvl += r->next->Data.level - r->Data.level;

    r = r->next;
  }

  return('0');
}
//---------------------------------------------------------------------------
--
void TForm1::SPIHTEncode()
{
  int tmplevel;
  long long int tmpthreshold;

  /*Height = Width = 8;
  AllocMemForLongIntMatrix(Y,Height,Width);
  Y[0][0] = 63;
  Y[0][1] = -34;
  Y[0][2] = 49;
  Y[0][3] = 10;
  Y[0][4] = 7;
  Y[0][5] = 13;
```

```
Y[0][6] = -12;
Y[0][7] = 7;
Y[1][0] = -31;
Y[1][1] = 23;
Y[1][2] = 14;
Y[1][3] = -13;
Y[1][4] = 3;
Y[1][5] = 4;
Y[1][6] = 6;
Y[1][7] = -1;
Y[2][0] = 15;
Y[2][1] = 14;
Y[2][2] = 3;
Y[2][3] = -12;
Y[2][4] = 5;
Y[2][5] = -7;
Y[2][6] = 3;
Y[2][7] = 9;
Y[3][0] = -9;
Y[3][1] = -7;
Y[3][2] = -14;
Y[3][3] = 8;
Y[3][4] = 4;
Y[3][5] = -2;
Y[3][6] = 3;
Y[3][7] = 2;
Y[4][0] = -5;
Y[4][1] = 9;
Y[4][2] = -1;
Y[4][3] = 47;
Y[4][4] = 4;
Y[4][5] = 6;
Y[4][6] = -2;
Y[4][7] = 2;
Y[5][0] = 3;
Y[5][1] = 0;
Y[5][2] = -3;
Y[5][3] = 2;
Y[5][4] = 3;
Y[5][5] = -2;
Y[5][6] = 0;
Y[5][7] = 4;
Y[6][0] = 2;
Y[6][1] = -3;
Y[6][2] = 6;
Y[6][3] = -4;
Y[6][4] = 3;
Y[6][5] = 6;
Y[6][6] = 3;
Y[6][7] = 6;
Y[7][0] = 5;
Y[7][1] = 11;
Y[7][2] = 5;
Y[7][3] = 6;
Y[7][4] = 0;
Y[7][5] = 3;
Y[7][6] = -4;
Y[7][7] = 4;*/

//Delete all lists
DeleteList(LIPfirst, LIPlast);
DeleteList(LSPfirst, LSPlast);
DeleteList(LISfirst, LISlast);
```

```
  SPIHTInitialize(Y, Height, Width, levelenc, thresholdenc);

  //In order to keep original level and threshold
  tmplevel = levelenc;
  tmpthreshold = thresholdenc;

  //Allocates memory for string lists
  AllocMemForStringMatrix(S, tmplevel+1);
  AllocMemForStringMatrix(B, tmplevel+1);

  //Print LIS, LIP and LSP captions to StringGrid
  PrintToStringGridListsCapt(Form2->StringGrid1);
  PrintToStringGridListsCapt(Form2->StringGrid2);
  PrintToStringGridListsCapt(Form2->StringGrid3);

  int k = 0;
  while (tmplevel >= 0)
  {
    k++;

    tmpthreshold = pow(2,tmplevel);

    SPIHTSortingPassEnc(Y, Height, Width, tmplevel, tmpthreshold, S);
    SPIHTRefinementPassEnc(Y, Height, Width,tmplevel, B);

    //Print LIP, LSP, LIS to TStringGrid
    PrintToStringGridLists(Y, LISfirst, k, Form2->StringGrid1);
    PrintToStringGridLists(Y, LIPfirst, k, Form2->StringGrid2);
    PrintToStringGridLists(Y, LSPfirst, k, Form2->StringGrid3);

    Form2->Memo1->Lines->Add("S("   +   IntToStr(tmplevel)   +   ")="   +
S[tmplevel].c_str());
    Form2->Memo2->Lines->Add("B("   +   IntToStr(tmplevel)   +   ")="   +
B[tmplevel].c_str());

    tmplevel--;
  }
}
//---------------------------------------------------------------------------
--
void TForm1::SPIHTInitialize(long long int **Y, int m, int n, int &lvl,
long long int &thr)
{
  //SPIHT initialization (fills LIP, LSP, LIS)
  TList List;

  //Find level
  lvl = FindMaxSpectrumElement(Y, m, n);

  //Find threshold
  thr = pow(2,lvl);

  //Initialize LIP
  List.x = 0;
  List.y = 0;
  List.type = ' ';
  List.level = lvl;
  //Create first LIP element
  CreateList(LIPfirst, LIPlast, List);
  //Fill LIP with elements: (0,1);(1,0);(1,1)
  List.x = 0;
  List.y = 1;
  List.type = ' ';
  List.level = lvl;
```

```
  AddElementToList(LIPfirst,LIPlast, List);
  List.x = 1;
  List.y = 0;
  List.type = ' ';
  List.level = lvl;
  AddElementToList(LIPfirst,LIPlast, List);
  List.x = 1;
  List.y = 1;
  List.type = ' ';
  List.level = lvl;
  AddElementToList(LIPfirst,LIPlast, List);

  //Initialize LIS
  List.x = 0;
  List.y = 1;
  List.type = 'A';
  List.level = lvl;
  CreateList(LISfirst, LISlast, List);
  List.x = 1;
  List.y = 0;
  List.type = 'A';
  List.level = lvl;
  AddElementToList(LISfirst,LISlast, List);
  List.x = 1;
  List.y = 1;
  List.type = 'A';
  List.level = lvl;
  AddElementToList(LISfirst,LISlast, List);

  //Initialize LSP (creates blank list)
  /*List.x = NULL;
  List.y = NULL;
  List.type = NULL;
  List.level = NULL;
  CreateList(LSPfirst, LSPlast, List);}*/
}
//---------------------------------------------------------------------------
--
void TForm1::SPIHTSortingPassEnc(long long int **Y, int m, int n, int lvl,
long long int thr, string *S)
{
  //Sorting pass. Significance map encoding.

  //Deals with LIP
  TList List;
  TListElement *r;
  bool l_exists;
  string tmpeil;

  r = LIPfirst;
  while (r != NULL)
  {
    if (fabs(Y[r->Data.x][r->Data.y]) >= thr)
    {
      tmpeil.push_back('1');
      //S[r->Data.x][r->Data.y] = '1';
      if (Y[r->Data.x][r->Data.y] >= 0)
      {
        tmpeil.push_back('1');
        //Sign[r->Data.x][r->Data.y] = '1';
      }
      else
      {
        tmpeil.push_back('0');
```

```
        //Sign[r->Data.x][r->Data.y] = '0';
      }
      List.x = r->Data.x;
      List.y = r->Data.y;
      List.type = ' ';
      List.level = lvl;
      //Adds element to LSP
      AddElementToList(LSPfirst,LSPlast,List);
      //Removes element from LIP
      RemoveElementFromList(LIPfirst,LIPlast,r);
    }
    else
    {
      tmpeil.push_back('0');
      //S[r->Data.x][r->Data.y] = '0';
    }
    r = r->next;
  }

  //Deals with LIS
  r = LISfirst;
  while (r != NULL)
  {
      if (r->Data.type == 'A')
      {
        tmpeil.push_back(FindDSignificanceEnc(Y,   m,   n,   r->Data.x,   r-
>Data.y, thr));
        //SD[r->Data.x][r->Data.y] = FindDSignificance(Y, Height, Width, r-
>Data.x, r->Data.y);

        if (tmpeil[tmpeil.size() - 1] == '1')  //takes last
        //if (SD[r->Data.x][r->Data.y] == '1')
        {
          //Deals with O set
          if ((2*r->Data.x+1 < m) && (2*r->Data.y+1 < n))
          {
            //First O element
            if (abs(Y[2*r->Data.x][2*r->Data.y]) >= thr)
            {
              tmpeil.push_back('1');
              //S[2*r->Data.x][2*r->Data.y] = '1';
              if (Y[2*r->Data.x][2*r->Data.y] >= 0)
                tmpeil.push_back('1');
                //Sign[2*r->Data.x][2*r->Data.y] = '1';
              else
                tmpeil.push_back('0');
                //Sign[2*r->Data.x][2*r->Data.y] = '0';
              List.x = 2*r->Data.x;
              List.y = 2*r->Data.y;
              List.type = ' ';
              List.level = lvl;
              //Adds element to LSP
              AddElementToList(LSPfirst,LSPlast,List);
            }
            else
            {
              tmpeil.push_back('0');
              //S[2*r->Data.x][2*r->Data.y] = '0';
              List.x = 2*r->Data.x;
              List.y = 2*r->Data.y;
              List.type = ' ';
              List.level = lvl;
              //Adds element to LIP
              AddElementToList(LIPfirst,LIPlast,List);
```

```
}

//Second O element
if (fabs(Y[2*r->Data.x][2*r->Data.y+1]) >= thr)
{
  tmpeil.push_back('1');
  //S[2*r->Data.x][2*r->Data.y+1] = '1';
  if (Y[2*r->Data.x][2*r->Data.y+1] >= 0)
    tmpeil.push_back('1');
    //Sign[2*r->Data.x][2*r->Data.y+1] = '1';
  else
    tmpeil.push_back('0');
    //Sign[2*r->Data.x][2*r->Data.y+1] = '0';
  List.x = 2*r->Data.x;
  List.y = 2*r->Data.y+1;
  List.type = ' ';
  List.level = lvl;
  //Adds element to LSP
  AddElementToList(LSPfirst,LSPlast,List);
}
else
{
  tmpeil.push_back('0');
  //S[2*r->Data.x][2*r->Data.y+1] = '0';
  List.x = 2*r->Data.x;
  List.y = 2*r->Data.y+1;
  List.type = ' ';
  List.level = lvl;
  //Adds element to LIP
  AddElementToList(LIPfirst,LIPlast,List);
}

//Third O element
if (fabs(Y[2*r->Data.x+1][2*r->Data.y]) >= thr)
{
  tmpeil.push_back('1');
  //S[2*r->Data.x+1][2*r->Data.y] = '1';
  if (Y[2*r->Data.x+1][2*r->Data.y] >= 0)
    tmpeil.push_back('1');
    //Sign[2*r->Data.x+1][2*r->Data.y] = '1';
  else
    tmpeil.push_back('0');
    //Sign[2*r->Data.x+1][2*r->Data.y] = '0';
  List.x = 2*r->Data.x+1;
  List.y = 2*r->Data.y;
  List.type = ' ';
  List.level = lvl;
  //Adds element to LSP
  AddElementToList(LSPfirst,LSPlast,List);
}
else
{
  tmpeil.push_back('0');
  //S[2*r->Data.x+1][2*r->Data.y] = '0';
  List.x = 2*r->Data.x+1;
  List.y = 2*r->Data.y;
  List.type = ' ';
  List.level = lvl;
  //Adds element to LIP
  AddElementToList(LIPfirst,LIPlast,List);
}

//Fourth O element
if (fabs(Y[2*r->Data.x+1][2*r->Data.y+1]) >= thr)
```

```
            {
              tmpeil.push_back('1');
              //S[2*r->Data.x+1][2*r->Data.y+1] = '1';
              if (Y[2*r->Data.x+1][2*r->Data.y+1] >= 0)
                tmpeil.push_back('1');
                //Sign[2*r->Data.x+1][2*r->Data.y+1] = '1';
              else
                tmpeil.push_back('0');
                //Sign[2*r->Data.x+1][2*r->Data.y+1] = '0';
              List.x = 2*r->Data.x+1;
              List.y = 2*r->Data.y+1;
              List.type = ' ';
              List.level = lvl;
              //Adds element to LSP
              AddElementToList(LSPfirst,LSPlast,List);
            }
            else
            {
              tmpeil.push_back('0');
              //S[2*r->Data.x+1][2*r->Data.y+1] = '0';
              List.x = 2*r->Data.x+1;
              List.y = 2*r->Data.y+1;
              List.type = ' ';
              List.level = lvl;
              //Adds element to LIP
              AddElementToList(LIPfirst,LIPlast,List);
            }

          }


          FindLSignificanceEnc(Y,   m,   n,   r->Data.x,   r->Data.y,   thr,
l_exists);
          if (l_exists)
          {
            List.x = r->Data.x;
            List.y = r->Data.y;
            List.type = 'B';
            List.level = lvl;
            //Adds element to LIS
            AddElementToList(LISfirst,LISlast,List);
            //Removes element from LIS
            RemoveElementFromList(LISfirst,LISlast,r);
          }
          else
            //Removes element from LIS
            RemoveElementFromList(LISfirst,LISlast,r);

        }
      }
      if (r->Data.type == 'B')
      {
        tmpeil.push_back(FindLSignificanceEnc(Y,   m,   n,   r->Data.x,   r-
>Data.y, thr, l_exists));
        //SL[r->Data.x][r->Data.y] = FindLSignificance(Y, Height, Width, r-
>Data.x, r->Data.y,l_exists);

        if ((tmpeil[tmpeil.size() - 1] == '1') && (l_exists))
        //if ((SL[r->Data.x][r->Data.y] == '1') && (l_exists))
        {
          //First O element
          List.x = 2*r->Data.x;
          List.y = 2*r->Data.y;
          List.type = 'A';
```

```
          List.level = lvl;
          //Adds element to LIS
          AddElementToList(LISfirst,LISlast,List);

          //Second O element
          List.x = 2*r->Data.x;
          List.y = 2*r->Data.y+1;
          List.type = 'A';
          List.level = lvl;
          //Adds element to LIS
          AddElementToList(LISfirst,LISlast,List);

          //Third O element
          List.x = 2*r->Data.x+1;
          List.y = 2*r->Data.y;
          List.type = 'A';
          List.level = lvl;
          //Adds element to LIS
          AddElementToList(LISfirst,LISlast,List);

          //Fourth O element
          List.x = 2*r->Data.x+1;
          List.y = 2*r->Data.y+1;
          List.type = 'A';
          List.level = lvl;
          //Adds element to LIS
          AddElementToList(LISfirst,LISlast,List);

          //Removes element from LIS
          RemoveElementFromList(LISfirst,LISlast,r);
        }
      }
  r = r->next;
  }

  S[lvl] = tmpeil;
  //Form2->Memo1->Lines->Add(tmpeil.c_str());
}
//---------------------------------------------------------------------------
--
void TForm1::SPIHTRefinementPassEnc(long long int **Y, int m, int n, int
lvl, string *B)
{
  TListElement *r;
  string tmpeil;

  r = LSPfirst;
  while ((r != NULL) && (lvl < r->Data.level))
  {
    //if (level < r->Data.level)

    tmpeil += FindMostSignificantBit(abs(Y[r->Data.x][r->Data.y]),lvl);
    r = r->next;
  }

  B[lvl] = tmpeil;
}
//---------------------------------------------------------------------------
--
void TForm1::SPIHTDecode()
{
  //SPIHT decoder
  int tmplevel;
  long long int tmpthreshold;
```

```
  int tmpuserlevel = StrToInt(Edit3->Text);;

  AllocMemForLongIntMatrix(XC, Height, Width);
  AllocMemForLongIntMatrix(YC, Height, Width);

  //Delete all lists
  DeleteList(LIPfirst, LIPlast);
  DeleteList(LSPfirst, LSPlast);
  DeleteList(LISfirst, LISlast);

  SPIHTInitialize(Y, Height, Width, levelenc, thresholdenc);

  tmplevel = levelenc;
  tmpthreshold = thresholdenc;

  //Print LIS, LIP and LSP captions to StringGrid
  PrintToStringGridListsCapt(Form3->StringGrid1);
  PrintToStringGridListsCapt(Form3->StringGrid2);
  PrintToStringGridListsCapt(Form3->StringGrid3);

  int k = 0;
  //while ((tmplevel >= 0) && (tmplevel >= tmpuserlevel)) //commented
12.05.15
  while ((tmplevel >= 0) && (tmplevel > levelenc - tmpuserlevel))
//12.05.15
  {
    k++;

    tmpthreshold = pow(2,tmplevel);

    SPIHTSortingPassDec(YC, Height, Width, S, tmplevel);
    SPIHTRefinementPassDec(YC, Height, Width, B, tmplevel);

    //Print LIP, LSP, LIS to TStringGrid
    PrintToStringGridLists(YC, LISfirst, k, Form3->StringGrid1);
    PrintToStringGridLists(YC, LIPfirst, k, Form3->StringGrid2);
    PrintToStringGridLists(YC, LSPfirst, k, Form3->StringGrid3);

    Form3->Memo1->Lines->Add("S(" + IntToStr(tmplevel) + ")=" +
S[tmplevel].c_str());
    Form3->Memo2->Lines->Add("B(" + IntToStr(tmplevel) + ")=" +
B[tmplevel].c_str());

    tmplevel--;
  }

  PrintToStringGrid(YC,Height,Width,StringGrid3);
  DrawToCanvas(YC,Height,Width,Image3);

  FindView(XC, YC, Height, Width, true);   // Transform Rows
  FindView(XC, YC, Width, Height, false);  // Transform Collumns

  PrintToStringGrid(XC,Height,Width,StringGrid6);
  DrawToCanvas(XC,Height,Width,Image6);
}
//---------------------------------------------------------------------
--
void TForm1::SPIHTSortingPassDec(long long int **&Y, int n, int m, string
*S, int lvl)
{
//Sorting pass. Significance map decoding.

  //Deals with LIP
```

```
TList List;
TListElement *r;
bool l_exists;
string tmpeil;

int k = 0;
long long int coeff = 1.5 * pow(2,lvl);

tmpeil = S[lvl];
r = LIPfirst;
while (r != NULL)
{
  if (tmpeil[k] == '1')
  {
    k++;
    List.x = r->Data.x;
    List.y = r->Data.y;
    List.type = ' ';
    List.level = lvl;
    //Adds element to LSP
    AddElementToList(LSPfirst,LSPlast,List);
    //Removes element from LIP
    RemoveElementFromList(LIPfirst,LIPlast,r);
    //Calcs and adds coefficient to spectrum matrix
    if (tmpeil[k] == '1')
      Y[List.x][List.y] = coeff;
    else
      Y[List.x][List.y] = -coeff;
    k++;
  }
  else
    k++;

  r = r->next;
}

//Deals with LIS
r = LISfirst;
while (r != NULL)
{
    if (r->Data.type == 'A')
    {
      if (tmpeil[k] == '1')
      {
        k++;
        if ((2*r->Data.x+1 < m) && (2*r->Data.y+1 < n))
        {
          //First O element
          if (tmpeil[k] == '1')
          {
            k++;
            List.x = 2*r->Data.x;
            List.y = 2*r->Data.y;
            List.type = ' ';
            List.level = lvl;
            //Adds element to LSP
            AddElementToList(LSPfirst,LSPlast,List);
            //Calcs and adds coefficient to spectrum matrix
            if (tmpeil[k] == '1')
              Y[List.x][List.y] = coeff;
            else
              Y[List.x][List.y] = -coeff;
            k++;
          }
```

```
else
{
  k++;
  List.x = 2*r->Data.x;
  List.y = 2*r->Data.y;
  List.type = ' ';
  List.level = lvl;
  //Adds element to LIP
  AddElementToList(LIPfirst,LIPlast,List);
}

//Second O element
if (tmpeil[k] == '1')
{
  k++;
  List.x = 2*r->Data.x;
  List.y = 2*r->Data.y+1;
  List.type = ' ';
  List.level = lvl;
  //Adds element to LSP
  AddElementToList(LSPfirst,LSPlast,List);
  //Calcs and adds coefficient to spectrum matrix
  if (tmpeil[k] == '1')
    Y[List.x][List.y] = coeff;
  else
    Y[List.x][List.y] = -coeff;
  k++;
}
else
{
  k++;
  //S[2*r->Data.x][2*r->Data.y+1] = '0';
  List.x = 2*r->Data.x;
  List.y = 2*r->Data.y+1;
  List.type = ' ';
  List.level = lvl;
  //Adds element to LIP
  AddElementToList(LIPfirst,LIPlast,List);
}

//Third O element
if (tmpeil[k] == '1')
{
  k++;
  List.x = 2*r->Data.x+1;
  List.y = 2*r->Data.y;
  List.type = ' ';
  List.level = lvl;
  //Adds element to LSP
  AddElementToList(LSPfirst,LSPlast,List);
  //Calcs and adds coefficient to spectrum matrix
  if (tmpeil[k] == '1')
    Y[List.x][List.y] = coeff;
  else
    Y[List.x][List.y] = -coeff;;
  k++;
}
else
{
  k++;
  List.x = 2*r->Data.x+1;
  List.y = 2*r->Data.y;
  List.type = ' ';
  List.level = lvl;
```

```
        //Adds element to LIP
        AddElementToList(LIPfirst,LIPlast,List);
      }

      //Fourth O element
      if (tmpeil[k] == '1')
      {
        k++;
        List.x = 2*r->Data.x+1;
        List.y = 2*r->Data.y+1;
        List.type = ' ';
        List.level = lvl;
        //Adds element to LSP
        AddElementToList(LSPfirst,LSPlast,List);
        //Calcs and adds coefficient to spectrum matrix
        if (tmpeil[k] == '1')
          Y[List.x][List.y] = coeff;
        else
          Y[List.x][List.y] = -coeff;
        k++;
      }
      else
      {
        k++;
        List.x = 2*r->Data.x+1;
        List.y = 2*r->Data.y+1;
        List.type = ' ';
        List.level = lvl;
        //Adds element to LIP
        AddElementToList(LIPfirst,LIPlast,List);
      }

    }

    //Checks if L exists
    if ((4*r->Data.x < m) && (4*r->Data.y < n))
    {
      List.x = r->Data.x;
      List.y = r->Data.y;
      List.type = 'B';
      List.level = lvl;
      //Adds element to LIS
      AddElementToList(LISfirst,LISlast,List);
      //Removes element from LIS
      RemoveElementFromList(LISfirst,LISlast,r);
    }
    else
      //Removes element from LIS
      RemoveElementFromList(LISfirst,LISlast,r);
  }
  else
    k++;
}
if (r->Data.type == 'B')
{
  if (tmpeil[k] == '1')
  {
    k++;

    //First O element
    List.x = 2*r->Data.x;
    List.y = 2*r->Data.y;
    List.type = 'A';
    List.level = lvl;
```

```
          //Adds element to LIS
          AddElementToList(LISfirst,LISlast,List);

          //Second O element
          List.x = 2*r->Data.x;
          List.y = 2*r->Data.y+1;
          List.type = 'A';
          List.level = lvl;
          //Adds element to LIS
          AddElementToList(LISfirst,LISlast,List);

          //Third O element
          List.x = 2*r->Data.x+1;
          List.y = 2*r->Data.y;
          List.type = 'A';
          List.level = lvl;
          //Adds element to LIS
          AddElementToList(LISfirst,LISlast,List);

          //Fourth O element
          List.x = 2*r->Data.x+1;
          List.y = 2*r->Data.y+1;
          List.type = 'A';
          List.level = lvl;
          //Adds element to LIS
          AddElementToList(LISfirst,LISlast,List);

          //Removes element from LIS
          RemoveElementFromList(LISfirst,LISlast,r);
        }
        else
          k++;
      }
  r = r->next;
  }
}
//---------------------------------------------------------------------------
--
void  TForm1::SPIHTRefinementPassDec(long  long  int  **&Y,  int  m,  int  n,
string *B, int lvl)
{
  TListElement *r;
  string tmpeil;
  int k = 0;
  int long long coeff;

  tmpeil = B[lvl];
  r = LSPfirst;
  while ((r != NULL) && (lvl < r->Data.level))
  {
    coeff = abs(Y[r->Data.x][r->Data.y]);
    if (tmpeil[k] == '1')
    {
      if (lvl > 0)
        coeff += pow(2,lvl-1);
      else
        coeff += 0; //Just for same structure as below
    }
    else
    {
      if (lvl > 0)
        coeff -= pow(2,lvl-1);
      else
        coeff -= 1;
```

```
    }

    if (Y[r->Data.x][r->Data.y] >= 0)
      Y[r->Data.x][r->Data.y] = coeff;
    else
      Y[r->Data.x][r->Data.y] = -coeff;

    k++;
    r = r->next;
  }
}
//(---SS120421)
//---------------------------------------------------------------------------
--
void __fastcall TForm1::TrackBar1Change(TObject *Sender)
{
  Label4->Caption = "2^" + IntToStr(TrackBar1->Position);
  int multiplier = pow(2,TrackBar1->Position);
  ChangeLuminance(Y, Height, Width, multiplier, StringGrid2, Image2);

  /*double **A;
  AllocMemForMatrix(A,Height,Width);
  Label4->Caption = "2^" + IntToStr(TrackBar1->Position);
  int multiplier = pow(2,TrackBar1->Position);
  for (int i = 0; i < Height; i++)
    for (int j = 0; j < Width; j++)
    {
      double absolute = fabs(Y[i][j]) * multiplier;
      if (absolute - 0.000001 < 0) {
        absolute = 0.000001;
        absolute = absolute * multiplier;
      }
      if (absolute > 255)
        absolute = 255;
      A[i][j] = absolute;
      Image2->Canvas->Pixels[j][i]                                        =
(TColor)RGB(absolute,absolute,absolute);
    }
      PrintToStringGrid(A,Height,Width,StringGrid2);
      //Image2->Canvas->Pixels[j][i]                                      =
(TColor)RGB(fabs(XfY[i][j])*daug,fabs(XfY[i][j])*daug,fabs(XfY[i][j])*daug)
;*/
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::Pasirinktinuotrauk11Click(TObject *Sender)
{
  if (OpenPictureDialog1->Execute())
  {
    Image1->Picture->LoadFromFile(OpenPictureDialog1->FileName);
  }
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::Spektr1Click(TObject *Sender)
{
  // Check if image is 8 bit
  assert(Image1->Picture->Bitmap->PixelFormat == pf8bit);

  // Get height and width of the image
  Height = Image1->Picture->Bitmap->Height;
  Width = Image1->Picture->Bitmap->Width;

  // Alocate mamory for view matrix
```

```
  AllocMemForLongIntMatrix(X,Height,Width);
  AllocMemForDoubleMatrix(XDbl,Height,Width); //(12.05.15)

  // Get pixels color from image and store to matrix
  for (int i = 0; i < Height; i++)
  {
    // Grab a pointer to the y-th row
    unsigned char const* const p_row =
      static_cast<unsigned char*>(Image1->Picture->Bitmap->ScanLine[i]);

    // For each column (pixel of the current row)
    for (int j = 0; j < Width; j++)
    {
      X[i][j] = p_row[j];
      XDbl[i][j] = p_row[j];  //(12.05.15)
    }
  }
  PrintToStringGrid(X,Height,Width,StringGrid1);
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::Spektr2Click(TObject *Sender)
{
  double starttime, endtime, duration;

  AllocMemForLongIntMatrix(Y,Height,Width);

  starttime = timeGetTime();

  // Find spectrum
  FindSpectrum(X, Y, Width, Height, false); // Transform Collumns
  FindSpectrum(X, Y, Height, Width, true);  // Transform Rows

  endtime = timeGetTime();
  duration = (endtime - starttime) / 1000;

  PrintToStringGrid(Y,Height,Width,StringGrid2);
  DrawToCanvas(Y, Height, Width, Image2);

  StatusBar1->SimplePanel = true;
  StatusBar1->SimpleText    =    "Operacijos    trukmė    (1)    "    +
FloatToStrF(duration,ffFixed,8,3) + " sekundės";

  //Alocate mamory for view from spectrum matrix
  //AllocMemForMatrix(XfY,Height,Width);

  // Find view from spectrum
  //FindView(XfY, Y, Height, Width, true);   // Transform Rows
  //FindView(XfY, Y, Width, Height, false);  // Transform Collumns

  //PrintToStringGrid(XfY,Height,Width,StringGrid3);

  //for (int i = 0; i < Height; i++)
    //for (int j = 0; j < Width; j++)
      //Image3->Canvas->Pixels[j][i]                                    =
(TColor)RGB(XfY[i][j],XfY[i][j],XfY[i][j]);
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::Fragmentivaizdo1Click(TObject *Sender)
{
  // Find fragment from view
  double starttime, endtime, duration;
  int lh, lw;
```

```
  int k1, k2;

  k1 = StrToInt(Edit1->Text);
  k2 = StrToInt(Edit2->Text);

  starttime = timeGetTime();

  AllocMemForDoubleMatrix(H1,Height,Height);
  AllocMemForDoubleMatrix(H2,Width,Width);
  //if (Height > Width)
    //FindHTMatrix(H,Height);
  //else
    //FindHTMatrix(H,Width);

  FindHTMatrix(H1,Height);
  FindHTMatrix(H2,Width);

  FindPartFromView(k1, k2, lh, lw);

  endtime = timeGetTime();
  duration = (endtime - starttime) / 1000;

  PrintToStringGrid(YfragFView,lh,lw,StringGrid4);
  DrawToCanvas(YfragFView,lh,lw,Image4);

  StatusBar1->SimplePanel = true;
  StatusBar1->SimpleText     =     "Operacijos     trukmė     (1)     "     +
FloatToStrF(duration,ffFixed,8,3) + " sekundės";

  q1 = lh;
  q2 = lw;
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::Fragmentispektro1Click(TObject *Sender)
{
  // Find fragment from spectrum

  int k1, k2, auk, plo;
  double starttime, endtime, duration;

  k1 = StrToInt(Edit1->Text);
  k2 = StrToInt(Edit2->Text);

  starttime = timeGetTime();

  //FindSpectralCoeffs(k1, k2, Height);
  FindSpectralCoeffsNewAlgorythm(k1, k2, auk, plo);

  endtime = timeGetTime();
  duration = (endtime - starttime) /1000;

  PrintToStringGridDouble(YfragFSpectrDbl, auk, plo, StringGrid8);
  DrawToCanvasDouble(YfragFSpectrDbl, auk, plo, Image8);

  StatusBar1->SimplePanel = true;
  StatusBar1->SimpleText     =     "Operacijos     trukmė     (2)     "     +
FloatToStrF(duration,ffFixed,8,3) + " sekundės";

  w1 = auk;
  w2 = plo;

  //Alocate mamory for view from spectrum matrix
  //AllocMemForMatrix(XfragFSpectr,Height,Width);
```

```
  //FindView(XfragFSpectr,  YfragFSpectr,  Height,  Width,  true);       //
Transform Rows
  //FindView(XfragFSpectr,  YfragFSpectr,  Width,  Height,  false);     //
Transform Collumns

  //PrintToStringGrid(XfragFSpectr, Height, Width, StringGrid5);

  //for (int i = 0; i < auk; i++)
    //for (int j = 0; j < plo; j++)
      //Image4->Canvas->Pixels[j][i]                                      =
(TColor)RGB(XfragFSpectr[i][j],XfragFSpectr[i][j],XfragFSpectr[i][j]);
      //Image4->Canvas->Pixels[j][i]                                      =
(TColor)RGB(fabs(YfragFSpectr[i][j])*pow(2,11),fabs(YfragFSpectr[i][j])*pow
(2,11),fabs(YfragFSpectr[i][j])*pow(2,11));
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if (SavePictureDialog1->Execute())
  {
    Image2->Picture->SaveToFile(SavePictureDialog1->FileName + ".bmp");
  }
}
//---------------------------------------------------------------------------
--

void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  if (SaveDialog1->Execute())
  {
    SaveToCSV(X, Height, Width, SaveDialog1->FileName + ".csv");
  }
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
  if (SaveDialog1->Execute())
  {
    SaveToCSV(Y, Height, Width, SaveDialog1->FileName + ".csv");
  }
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
  if (SaveDialog1->Execute())
  {
    SaveToCSV(YfragFView, q1, q2, SaveDialog1->FileName + ".csv");
  }
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{
  if (SaveDialog1->Execute())
  {
    SaveToCSV(YfragFSpectr, w1, w2, SaveDialog1->FileName + ".csv");
  }
}
//---------------------------------------------------------------------------
--
```

```
void __fastcall TForm1::BitBtn6Click(TObject *Sender)
{
  if (SavePictureDialog1->Execute())
  {
    Image3->Picture->SaveToFile(SavePictureDialog1->FileName + ".bmp");
  }
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::BitBtn7Click(TObject *Sender)
{
  if (SavePictureDialog1->Execute())
  {
    Image4->Picture->SaveToFile(SavePictureDialog1->FileName + ".bmp");
  }
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::TrackBar2Change(TObject *Sender)
{
  Label12->Caption = "2^" + IntToStr(TrackBar2->Position);
  int multiplier = pow(2,TrackBar2->Position);
  ChangeLuminance(YC, Height, Width, multiplier, StringGrid3, Image3);
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::TrackBar3Change(TObject *Sender)
{
  Label13->Caption = "2^" + IntToStr(TrackBar3->Position);
  int multiplier = pow(2,TrackBar3->Position);
  ChangeLuminance(YfragFView, q1, q1, multiplier, StringGrid4, Image4);
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::TrackBar4Change(TObject *Sender)
{
  Label16->Caption = "2^" + IntToStr(TrackBar4->Position);
  int multiplier = pow(2,TrackBar4->Position);
  ChangeLuminance(YfragFSpectr, w1, w1, multiplier, StringGrid8, Image8);
}
//---------------------------------------------------------------------------
--

void __fastcall TForm1::Testinmatrica1Click(TObject *Sender)
{
//(12.03.10 for testing purpouses)
Height = 8;
Width = 8;

//uzkraunam matrica
AllocMemForLongIntMatrix(X,Height,Width);

X[0][0] = 63;
X[0][1] = -34;
X[0][2] = 49;
X[0][3] = 10;
X[0][4] = 7;
X[0][5] = 13;
X[0][6] = -12;
X[0][7] = 7;
X[1][0] = -31;
X[1][1] = 23;
X[1][2] = 14;
X[1][3] = -13;
```

```
X[1][4] = 3;
X[1][5] = 4;
X[1][6] = 6;
X[1][7] = -1;
X[2][0] = 15;
X[2][1] = 14;
X[2][2] = 3;
X[2][3] = -12;
X[2][4] = 5;
X[2][5] = -7;
X[2][6] = 3;
X[2][7] = 9;
X[3][0] = -9;
X[3][1] = -7;
X[3][2] = -14;
X[3][3] = 8;
X[3][4] = 4;
X[3][5] = -2;
X[3][6] = 3;
X[3][7] = 2;
X[4][0] = -5;
X[4][1] = 9;
X[4][2] = -1;
X[4][3] = 47;
X[4][4] = 4;
X[4][5] = 6;
X[4][6] = -2;
X[4][7] = 2;
X[5][0] = 3;
X[5][1] = 0;
X[5][2] = -3;
X[5][3] = 2;
X[5][4] = 3;
X[5][5] = -2;
X[5][6] = 0;
X[5][7] = 4;
X[6][0] = 2;
X[6][1] = -3;
X[6][2] = 6;
X[6][3] = -4;
X[6][4] = 3;
X[6][5] = 6;
X[6][6] = 3;
X[6][7] = 6;
X[7][0] = 5;
X[7][1] = 11;
X[7][2] = 5;
X[7][3] = 6;
X[7][4] = 0;
X[7][5] = 3;
X[7][6] = -4;
X[7][7] = 4;

PrintToStringGrid(X,Height,Width,StringGrid1);


//randam spektra
AllocMemForLongIntMatrix(Y,Height,Width);

  // Find spectrum
FindSpectrum(X, Y, Width, Height, false); // Transform Collumns
FindSpectrum(X, Y, Height, Width, true);  // Transform Rows

PrintToStringGrid(Y,Height,Width,StringGrid2);
```

```
//randam vaizda is spektro su atvirkstine transformacija
AllocMemForLongIntMatrix(XfY,Height,Width);

FindView(XfY, Y, Height, Width, true);   // Transform Rows
FindView(XfY, Y, Width, Height, false);  // Transform Collumns

PrintToStringGrid(XfY,Height,Width,StringGrid6);
}
//---------------------------------------------------------------------------
--


void __fastcall TForm1::SPIHTtest1Click(TObject *Sender)
{
  /*Height = Width = 4;
  AllocMemForMatrix(Y,Height,Width);

  Y[0][0] = 26;
  Y[0][1] = 6;
  Y[0][2] = 13;
  Y[0][3] = 10;
  Y[1][0] = -7;
  Y[1][1] = 7;
  Y[1][2] = 6;
  Y[1][3] = 4;
  Y[2][0] = 4;
  Y[2][1] = -4;
  Y[2][2] = 4;
  Y[2][3] = -3;
  Y[3][0] = 2;
  Y[3][1] = -2;
  Y[3][2] = -2;
  Y[3][3] = 0;*/

  /*Height = Width = 8;
  AllocMemForLongIntMatrix(Y,Height,Width);
  Y[0][0] = 63;
  Y[0][1] = -34;
  Y[0][2] = 49;
  Y[0][3] = 10;
  Y[0][4] = 7;
  Y[0][5] = 13;
  Y[0][6] = -12;
  Y[0][7] = 7;
  Y[1][0] = -31;
  Y[1][1] = 23;
  Y[1][2] = 14;
  Y[1][3] = -13;
  Y[1][4] = 3;
  Y[1][5] = 4;
  Y[1][6] = 6;
  Y[1][7] = -1;
  Y[2][0] = 15;
  Y[2][1] = 14;
  Y[2][2] = 3;
  Y[2][3] = -12;
  Y[2][4] = 5;
  Y[2][5] = -7;
  Y[2][6] = 3;
  Y[2][7] = 9;
  Y[3][0] = -9;
  Y[3][1] = -7;
  Y[3][2] = -14;
```

```
Y[3][3] = 8;
Y[3][4] = 4;
Y[3][5] = -2;
Y[3][6] = 3;
Y[3][7] = 2;
Y[4][0] = -5;
Y[4][1] = 9;
Y[4][2] = -1;
Y[4][3] = 47;
Y[4][4] = 4;
Y[4][5] = 6;
Y[4][6] = -2;
Y[4][7] = 2;
Y[5][0] = 3;
Y[5][1] = 0;
Y[5][2] = -3;
Y[5][3] = 2;
Y[5][4] = 3;
Y[5][5] = -2;
Y[5][6] = 0;
Y[5][7] = 4;
Y[6][0] = 2;
Y[6][1] = -3;
Y[6][2] = 6;
Y[6][3] = -4;
Y[6][4] = 3;
Y[6][5] = 6;
Y[6][6] = 3;
Y[6][7] = 6;
Y[7][0] = 5;
Y[7][1] = 11;
Y[7][2] = 5;
Y[7][3] = 6;
Y[7][4] = 0;
Y[7][5] = 3;
Y[7][6] = -4;
Y[7][7] = 4;

//SPIHTInitialize();

AllocMemForStringMatrix(S, level+1);
AllocMemForStringMatrix(B, level+1);

TListElement *r;

Form2->StringGrid1->Cells[0][0] = "X";
Form2->StringGrid1->Cells[1][0] = "Y";
Form2->StringGrid1->Cells[2][0] = "TYPE";
Form2->StringGrid1->Cells[3][0] = "LEVEL";
Form2->StringGrid1->Cells[4][0] = "VALUE";
Form2->StringGrid2->Cells[0][0] = "X";
Form2->StringGrid2->Cells[1][0] = "Y";
Form2->StringGrid2->Cells[2][0] = "TYPE";
Form2->StringGrid2->Cells[3][0] = "LEVEL";
Form2->StringGrid2->Cells[4][0] = "VALUE";
Form2->StringGrid3->Cells[0][0] = "X";
Form2->StringGrid3->Cells[1][0] = "Y";
Form2->StringGrid3->Cells[2][0] = "TYPE";
Form2->StringGrid3->Cells[3][0] = "LEVEL";
Form2->StringGrid3->Cells[4][0] = "VALUE";

int j1 = 0;
int j2 = 0;
int j3 = 0;
```

```
Form2->StringGrid1->ColCount = 6;
Form2->StringGrid1->RowCount = 0;
Form2->StringGrid2->ColCount = 6;
Form2->StringGrid2->RowCount = 0;
Form2->StringGrid3->ColCount = 6;
Form2->StringGrid3->RowCount = 0;

int k = 0;
for (int i = level; i >= 0; i--)
{
  k++;

  SPIHTSortingPassEnc(Y);
  SPIHTRefinementPassEnc(Y);

  r = LSPfirst;
  while (r != NULL)
  {
    j1++;
    Form2->StringGrid1->Cells[0][j1] = r->Data.x;
    Form2->StringGrid1->Cells[1][j1] = r->Data.y;
    Form2->StringGrid1->Cells[2][j1] = r->Data.type;
    Form2->StringGrid1->Cells[3][j1] = r->Data.level;
    Form2->StringGrid1->Cells[4][j1] = Y[r->Data.x][r->Data.y];
    Form2->StringGrid1->Cells[5][j1] = k;
    Form2->StringGrid1->RowCount += 1;
    r = r->next;
  }

  r = LIPfirst;
  while (r != NULL)
  {
    j2++;
    Form2->StringGrid2->Cells[0][j2] = r->Data.x;
    Form2->StringGrid2->Cells[1][j2] = r->Data.y;
    Form2->StringGrid2->Cells[2][j2] = r->Data.type;
    Form2->StringGrid2->Cells[3][j2] = r->Data.level;
    Form2->StringGrid2->Cells[4][j2] = Y[r->Data.x][r->Data.y];
    Form2->StringGrid2->Cells[5][j2] = k;
    Form2->StringGrid2->RowCount += 1;
    r = r->next;
  }

  r = LISfirst;
  while (r != NULL)
  {
    j3++;
    Form2->StringGrid3->Cells[0][j3] = r->Data.x;
    Form2->StringGrid3->Cells[1][j3] = r->Data.y;
    Form2->StringGrid3->Cells[2][j3] = r->Data.type;
    Form2->StringGrid3->Cells[3][j3] = r->Data.level;
    Form2->StringGrid3->Cells[4][j3] = Y[r->Data.x][r->Data.y];
    Form2->StringGrid3->Cells[5][j3] = k;
    Form2->StringGrid3->RowCount += 1;
    r = r->next;
  }

  Form2->Memo1->Lines->Add(S[level].c_str());
  Form2->Memo2->Lines->Add(B[level].c_str());
level--;
threshold = pow(2,level);
}

//Decode//////////////////////////////////////////
```

```
  AllocMemForLongIntMatrix(YC, Height, Width);
  SPIHTInitialize();
  for (int i = level; i >= 0; i--)
  {
    SPIHTSortingPassDec(YC, S);
    SPIHTRefinementPassDec(YC, B);
    level--;
  }

  StringGrid1->RowCount = Height+1;
  StringGrid1->ColCount = Width+1;
  for (int i = 0; i < Height; i++)
    for (int j = 0; j < Width; j++)
      StringGrid1->Cells[j+1][i+1] = YC[i][j]; */
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form2->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  Form3->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Koduoti1Click(TObject *Sender)
{
  double starttime, endtime, duration;
  starttime = timeGetTime();
  SPIHTEncode();
  endtime = timeGetTime();
  duration = (endtime - starttime) / 1000;
  StatusBar1->SimplePanel = true;
  StatusBar1->SimpleText     =     "Operacijos     trukmė     (1)     "     +
FloatToStrF(duration,ffFixed,8,3) + " sekundės";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Dekoduoti1Click(TObject *Sender)
{
  double starttime, endtime, duration;
  starttime = timeGetTime();
  SPIHTDecode();
  endtime = timeGetTime();
  duration = (endtime - starttime) / 1000;
  StatusBar1->SimplePanel = true;
  StatusBar1->SimpleText     =     "Operacijos     trukmė     (2)     "     +
FloatToStrF(duration,ffFixed,8,3) + " sekundės";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Kodavimoprocesas1Click(TObject *Sender)
{
  Form2->Show();
}
```

```
//---------------------------------------------------------------------------
--
void __fastcall TForm1::Dekodavimoprocesas1Click(TObject *Sender)
{
  Form3->Show();
}
//---------------------------------------------------------------------------
--

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Height = Width = 8;
  AllocMemForLongIntMatrix(X,Height,Width);
  X[0][0] = 63;
  X[0][1] = -34;
  X[0][2] = 49;
  X[0][3] = 10;
  X[0][4] = 7;
  X[0][5] = 13;
  X[0][6] = -12;
  X[0][7] = 7;
  X[1][0] = -31;
  X[1][1] = 23;
  X[1][2] = 14;
  X[1][3] = -13;
  X[1][4] = 3;
  X[1][5] = 4;
  X[1][6] = 6;
  X[1][7] = -1;
  X[2][0] = 15;
  X[2][1] = 14;
  X[2][2] = 3;
  X[2][3] = -12;
  X[2][4] = 5;
  X[2][5] = -7;
  X[2][6] = 3;
  X[2][7] = 9;
  X[3][0] = -9;
  X[3][1] = -7;
  X[3][2] = -14;
  X[3][3] = 8;
  X[3][4] = 4;
  X[3][5] = -2;
  X[3][6] = 3;
  X[3][7] = 2;
  X[4][0] = -5;
  X[4][1] = 9;
  X[4][2] = -1;
  X[4][3] = 47;
  X[4][4] = 4;
  X[4][5] = 6;
  X[4][6] = -2;
  X[4][7] = 2;
  X[5][0] = 3;
  X[5][1] = 0;
  X[5][2] = -3;
  X[5][3] = 2;
  X[5][4] = 3;
  X[5][5] = -2;
  X[5][6] = 0;
  X[5][7] = 4;
  X[6][0] = 2;
  X[6][1] = -3;
  X[6][2] = 6;
```

```
  X[6][3] = -4;
  X[6][4] = 3;
  X[6][5] = 6;
  X[6][6] = 3;
  X[6][7] = 6;
  X[7][0] = 5;
  X[7][1] = 11;
  X[7][2] = 5;
  X[7][3] = 6;
  X[7][4] = 0;
  X[7][5] = 3;
  X[7][6] = -4;
  X[7][7] = 4;

  PrintToStringGrid(X,Height,Width,StringGrid1);
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::Rastibendrjvaizd1Click(TObject *Sender)
{
  AllocMemForLongIntMatrix(XTot,Height,Width);

  //Add compressed image to view
  for (int i = 0; i < Height; i++)
    for (int j = 0; j < Width; j++)
      XTot[i][j] = XC[i][j];

  //Add fine quality fragment to view
  for (int i = 0; i < q1; i++)
    for (int j = 0; j < q2; j++)
      XTot[StartFragmentCordX+i][StartFragmentCordY+j] = XfragFView[i][j];

  //Find total spectrum
  AllocMemForLongIntMatrix(YTot,Height,Width);
  FindSpectrum(XTot, YTot, Width, Height, false); // Transform Collumns
  FindSpectrum(XTot, YTot, Height, Width, true);  // Transform Rows

  //Calculate MSE
  double MSE = 0;
  for (int i = 0; i < Width; i++)
    for (int j = 0; j < Height; j++)
      MSE += pow(XTot[i][j]-X[i][j],2);
  MSE /= (Height*Width);

  Label20->Caption = "MSE=" + FloatToStrF(MSE,ffFixed,8,2);

  //Print total view
  PrintToStringGrid(XTot,Height,Width,StringGrid9);
  DrawToCanvas(XTot,Height,Width,Image9);

  //Print total spectrum
  PrintToStringGrid(YTot,Height,Width,StringGrid10);
  DrawToCanvas(YTot,Height,Width,Image10);
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::Rastirealijskaiispektr1Click(TObject *Sender)
{
  double starttime, endtime, duration;

  AllocMemForDoubleMatrix(YDbl,Height,Width);

  starttime = timeGetTime();
```

```
  // Find spectrum
  FindSpectrumDouble(XDbl,  YDbl,  Width,  Height,  false);  //  Transform
Collumns
  FindSpectrumDouble(XDbl, YDbl, Height, Width, true);  // Transform Rows

  endtime = timeGetTime();
  duration = (endtime - starttime) / 1000;

  PrintToStringGridDouble(YDbl,Height,Width,StringGrid5);
  DrawToCanvasDouble(YDbl, Height, Width, Image5);

  StatusBar1->SimplePanel = true;
  StatusBar1->SimpleText    =    "Operacijos    trukmė    (2)    "    +
FloatToStrF(duration,ffFixed,8,3) + " sekundės";
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::Rastifragmentivaizdorealijsk1Click(TObject *Sender)
{
// Find fragment from view
  double starttime, endtime, duration;
  int lh, lw;
  int k1, k2;

  k1 = StrToInt(Edit1->Text);
  k2 = StrToInt(Edit2->Text);

  starttime = timeGetTime();

  AllocMemForDoubleMatrix(H1,Height,Height);
  AllocMemForDoubleMatrix(H2,Width,Width);
  //if (Height > Width)
    //FindHTMatrix(H,Height);
  //else
    //FindHTMatrix(H,Width);

  FindHTMatrix(H1,Height);
  FindHTMatrix(H2,Width);

  FindPartFromViewDouble(k1, k2, lh, lw);

  endtime = timeGetTime();
  duration = (endtime - starttime) / 1000;

  PrintToStringGridDouble(YfragFViewDbl,lh,lw,StringGrid4);
  DrawToCanvasDouble(YfragFViewDbl,lh,lw,Image4);

  StatusBar1->SimplePanel = true;
  StatusBar1->SimpleText    =    "Operacijos    trukmė    (1)    "    +
FloatToStrF(duration,ffFixed,8,3) + " sekundės";

  q1 = lh;
  q2 = lw;
}
//---------------------------------------------------------------------------
--
void __fastcall TForm1::BitBtn15Click(TObject *Sender)
{
  if (SavePictureDialog1->Execute())
  {
    Image9->Picture->SaveToFile(SavePictureDialog1->FileName + ".bmp");
  }
}
//---------------------------------------------------------------------------
```