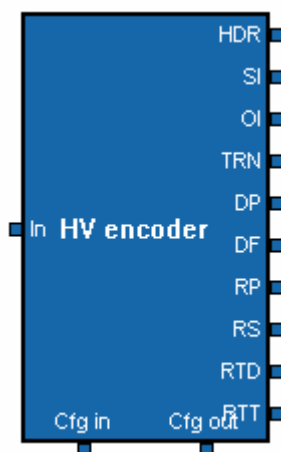


# HV encoder

## Obsah

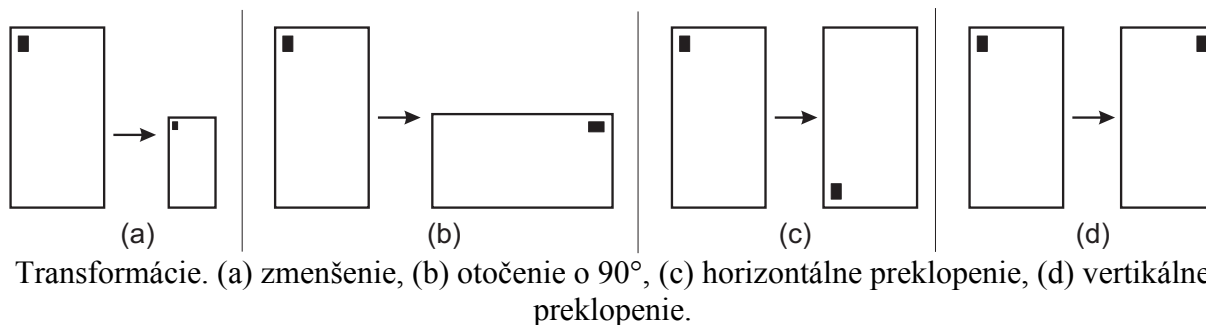
1. Popis.....	1
2. Závislosti .....	2
3. Implementačné informácie .....	3
4. Piny .....	3
5. Konfigurácia .....	4
5.1 Konfiguračná štruktúra.....	4
5.2 Konfiguračný dialóg.....	8
6. Definícia triedy .....	9
7. Príklad použitia .....	11

## 1. Popis

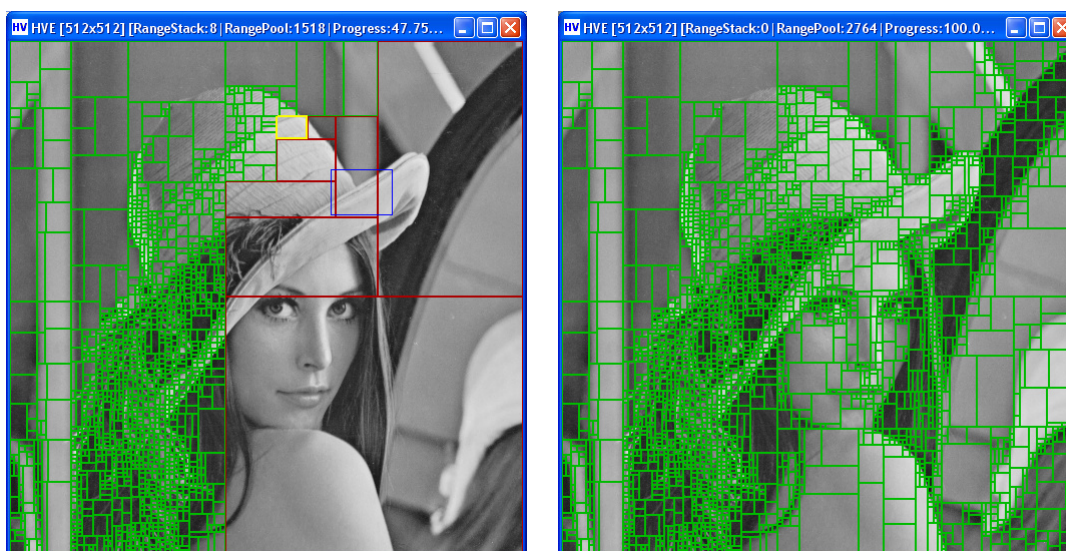


Trieda *HV encoder* je implementáciou horizontálno-vertikálnej schémy fraktálnej transformácie. Vstupom je šedo tónový obraz (v ľubovoľnom dynamickom rozsahu) v štruktúre *TPlane* a výstupom je množina afinných transformácií popisujúca IFS systém, ktorého atraktor je podobný vstupnému obrazu. Mieru kvality je možné nastaviť v konfiurácii.

*HV encoder* je kvôli časovej náročnosti prehľadávania obmedzený len na afinné transformácie. HV schéma rozdeľuje obraz na obdĺžnikové bloky. Kvôli tomu je množina afinných transformácie ešte zúžená len na určité vhodné typy. Uvažujeme len tieto typy afinných transformácií: posunutie, škálovanie (kvôli kontraktívnosti len zmenšenie), preklopenie okolo horizontálnej osi (horizontal flip), preklopenie okolo vertikálnej osi (vertical flip), otočenie o 90° doprava a ich kombinácie. Zložením týchto transformácií môžeme doceliť otočenie aj o iné uhly (180°, 270°). Transformácia je teda určená blokom (pozícia a rozmery), doménou (pozícia a faktory zväčšenia šírky a výšky vzhľadom na veľkosť bloku) a modifikáciou domény (otočenie o 90°, horizontálne a vertikálne preklopenie). Pozícia bloku a domény určuje posúvaciu časť transformácie, faktory zväčšenia šírky a výšky určujú škálovaciu časť transformácie, modifikáciu domény treba uložiť osobitne.



Modifikáciu domény zakódujeme tromi bitmi: 1 bit pre otočenie o 90°, 1 bit pre horizontálne preklopenie a 1 bit pre vertikálne preklopenie. Posúvaciu a škálovaciu časť transformácie budeme kódovať iným spôsobom. Povolením vizualizačného okna sa otvorí okno, v ktorom môžeme sledovať priebeh pokrývania.



V okne sa zobrazuje predloha (na pozadí), aktuálny strom blokov. Zelenou farbou sa zobrazujú pokryté bloky, červenou farbou nepokryté bloky a žltou farbou aktuálne pokrývaný blok. V záhlaví okna sa zobrazujú štatistické informácie ako rozlíšenie vstupu, počet nepokrytých blokov v zásobníku (*RangeStack*), počet pokrytých blokov (*RangePool*), percentuálna kompletnosť operácie (*Progress*) a indikátor korektného pokrytia (*CC – Corect Cover*, ak je hodnota *TRUE*, pokrytie je zatiaľ korektné, ak *FALSE*, algoritmus bol donútený akceptovať pokrytie bloku, ktoré nevyhovuje zadaným podmienkam, napr. blok bolo treba rozdeliť ale pre malé rozmery to nebolo umožnené).

## 2. Závislosti

Trieda *HV encoder* používa tieto externé definičné súbory a knižnice:

**Headers:** *filtergraph.h*, *datatypes.h*

**Libs:** *filtergraph.lib*

### 3. Implementačné informácie

Informácie o triede:

**Názov:** *HV encoder*

**Verzia:** *1.0*

**Magic:** *130*

Informácie o definíciách:

**Header:** *hvencoder.h*

**Konfiguračný header:** *hvencodercfg.h*

**Lib:** *hvencoder.lib*

Informácie o knižnici obsahujúcej triedu:

**Názov knižnice:** *HV encoder Library*

**Verzia knižnice:** *1.0*

**Dll súbor knižnice:** *hvencoder.dll*

### 4. Piny

Piny sú popísané spôsobom: „*[index pinu na filtri] názov pinu: popis pinu*“.

**[0] Cfg in:** Konfiguračný vstup. Vstupom je serializovaná konfigurácia.

**[1] Cfg out:** Konfiguračný výstup. Výstupom je serializovaná konfigurácia.

**[2] In:** Vstup. Vstupom je šedo tónový obraz v štruktúre *TPlane*.

**[3] HDR:** Výstup *Header*. Hlavičková štruktúra obsahujúca informácia potrebné na korektné dekódovanie.

**[4] SI:** Výstup *Scale Indices*. Kvantizačné indexy škálovacích faktorov domén, kódované ako pole celých čísel v štruktúre *TIntArray*.

**[5] OI:** Výstup *Offset Indices*. Kvantizačné indexy posúvacích faktorov domén, kódované ako pole celých čísel v štruktúre *TIntArray*.

**[6] TRN:** Výstup *Transformations*. Číselne zakódované modifikačné časti transformácií, kódované ako pole celých čísel v štruktúre *TIntArray*.

**[7] DP:** Výstup *Domain Positions*. Pozície domén predelené hustotou prehľadávacej mriežky, kódované po dvojiciach ako pole celých čísel v štruktúre *TIntArray*.

**[8] DF:** Výstup *Domain Factors*. Škálovacie faktory blokov, kódované po dvojiciach ako pole celých čísel v štruktúre *TIntArray*.

**[9] RP:** Výstup *Range Positions*. Pozície blokov, kódované po dvojiciach ako pole celých čísel v štruktúre *TIntArray*.

**[10] RS:** Výstup *Range Sizes*. Veľkosti blokov, kódované po dvojiciach ako pole celých čísel v štruktúre *TIntArray*.

**[11] RTD:** Výstup *Range Tree Distanties*, kódované traverzom stromu blokov ako pole celých čísel v štruktúre *TIntArray*.

**[12] RTT:** Výstup *Range Tree Types*, kódované traverzom stromu blokov ako pole celých čísel v štruktúre *TIntArray*.

Všetky výstupné polia celých čísel sú synchronizované, teda paralelným prechodom cez všetky polia dostaneme prislúchajúce informácie o transformáciách. Napr. transformáciu *i* dostaneme zložením informácií z *SI[i]*, *OI[i]*, *TRN[i]*, *DP[2i]*, *DP[2i+1]*, *DF[2i]*, *DF[2i+1]*, *RP[2i]*, *RP[2i+1]*, *RS[2i]*, *RS[2i+1]*. Dvojica výstupných polí (*RP*, *RS*) obsahuje rovnaké informácie ako dvojica (*RTD*, *RTT*), ktoré sú však zapísané rôznou formou. Dvojica (*RP*, *RS*)

kóduje pozície a veľkosti blokov priamo. Dvojica (RTD, RTT) ich kóduje stromom, ktorý sa vytváral pri výpočte. Pozície a veľkosti blokov môžeme rekonštruovať z týchto údajov tak, že znovu vytvoríme strom blokov a vyberieme z neho listové vrcholy. Tieto dve dvojice poskytujú redundantné informácie a preto je vhodné kódovať len jednu z nich. Prvky z dvojice (RP, RS) majú približne rovnaký charakter. Naopak prvky z poľa RTD (vzdialenosti) majú iný charakter ako prvky poľa RTT (typy – čísla 0 a 1). Dvojicu (RTD, RTT) je možné efektívnejšie kódovať.

## 5. Konfigurácia

### 5.1 Konfiguračná štruktúra

Konfiguračná štruktúra je definovaná nasledovne:

```
typedef struct {
    int      CompressionMode;      // mód kompresie
    float     MaxError;             // maximálna povolená chyba
    BOOL      MaxDifCheck;         // indikátor kontroly maximálnej odchýlky
    int       MaxNumRanges;        // maximálny počet blokov
    int       MinRangeSize;        // minimálny rozmer bloku
    int       MaxRangeSize;        // maximálny rozmer bloku
    int       MinRangeFactor;      // minimálny faktor blok-doména
    int       MaxRangeFactor;      // maximálny faktor blok-doména
    float     MaxScaleFactor;      // maximálny škálovací faktor domény
    int       OffsetQLevels;       // počet kvantizačných úrovní pre posunutie
    int       ScaleQLevels;        // počet kvantizačných úrovní pre škálovanie
    int       SearchMode;          // prehľadávací mód
    int       GridSearch_StepX;    // veľkosť kroku v x-ovom smere
    int       GridSearch_StepY;    // veľkosť kroku v y-ovom smere
    int       CoveringMode;        // pokrývací mód
    int       ErrorMode;           // chybový mód
    float     Quality;             // kvalita
    int       NumThreads;          // počet paralelných výpočtových vlákien
    int       ThreadPriority;       // systémová priorita výpočtových vlákien
    BOOL      FeedbackWindow;      // vizualizačné okno
} THVEncoderConfig;

// módy kompresie
CM_LOSSY_FIDELITY    0    // stratový mód, cieľová vernosť
CM_LOSSY_SIZE        1    // stratový mód, cieľová veľkosť

// módy prehľadávania
SM_GRIDSEARCH    0    // mriežkové prehľadávanie

// módy pokrývania
CM_BESTCOVER        0    // hľadanie najlepšieho pokrytia
CM_COVERINTOLERANCE 1    // hľadanie pokrytia v danej tolerancii

// módy chyby
EM_ABSOLUTE        0    // absolútny mód
```

<i>EM_QUALITY</i>	<i>1</i>	// mód kvality
// systémové priority výpočtových vlákien		
<i>TP_NORMAL</i>	<i>0</i>	// normálna priorita
<i>TP_IDLE</i>	<i>1</i>	// nízka priorita, využívanie len voľných prostriedkov

Mód kompresie alebo voľba kompresného algoritmu. Tu máme na výber či je našim cieľom dosiahnuť stanovenú vernosť (mód *CM\_LOSSY\_FIDELITY*, *Fidelity*) alebo veľkosť (mód *CM\_LOSSY\_SIZE*, *Size*). Mód *Fidelity* neprihliada na množstvo transformácií, ktoré generuje ale snaží sa nájsť pokrytie v nastavenej tolerancii. Mód *Size* hľadá najlepšie pokrytie s použitím ohraničeného počtu blokov (transformácií), čím zabezpečí, že veľkosť výstupu bude zhora ohraničená.

Nastavenia módu *Fidelity*:

Mód chyby (*ErrorMode*): algoritmus používa RMS metriku na porovnanie podobnosti medzi pokrývaným blokom a zmenšenou doménou. Voľba *EM\_ABSOLUTE* nastaví priamo hodnotu maximálnej povolenej RMS odchýlky (alebo chyby) na hodnotu *MaxError*. Pri voľbe *EM\_QUALITY* si algoritmus vypočíta sám maximálnu povolenú odchýlku z dynamického rozsahu vstupného obrazu. Hodnotou *Quality* môžeme túto toleranciu ovplyvniť. Hodnota *1.0* znamená maximálnu kvalitu (žiadna tolerancia, akceptovaná len RMS odchýlka *0.0* t.j. žiadna chyba), naopak hodnota *0.0* znamená najnižšiu kvalitu. Väčšie hodnoty *Quality* spôsobia generovanie väčšieho množstva blokov a tým väčšiu vernosť ale aj veľkosť výstupu.

Mód pokrývania (*CoveringMode*). Voľba *CM\_BESTCOVER* dovoľuje algoritmu pokračovať a dokončiť prehľadávanie aj keď už bolo nájdené pokrytie v danej tolerancii. Algoritmus sa snaží nájsť najlepšie pokrytie bloku aj za cenu väčšej časovej náročnosti. Voľba *CM\_COVERINTOLERANCE* túto možnosť nepovoľuje a prehľadávanie sa zastaví pri nájdení prvého pokrytia v danej tolerancii.

Kontrola maximálnej odchýlky (*MaxDifCheck*) hovorí algoritmu aby udržiaval tiež maximálnu odchýlku bloku v tolerancii (nie len RMS odchýlku). Týmto je udržiavaná vyššia vernosť (väčšinou za cenu zväčšenia veľkosti výstupu). Tento parameter je zatiaľ len v experimentálnom stave a nie je zaručená 100%-ná funkčnosť.

Nastavenia módu *Size*:

Maximálny povolený počet blokov (*MaxNumRanges*): po dosiahnutí nastaveného počtu blokov sa algoritmus zastaví.

Mód prehľadávania (*SearchMode*) určuje akým spôsobom sa budú vyberať domény pre bloky. Momentálne je k dispozícii len voľba *SM\_GRIDSEARCH*. V móde *Grid search* sú domény vyberané zaradom z pozícií v mriežke s nastavenou hustotou každých *GridSearch\_StepX* pixelov v horizontálnom smere a *GridSearch\_StepY* pixelov vo vertikálnom smere. Tento spôsob prehľadávania je časovo náročný.

Nastavenia blokov:

Nastavenia obmedzujúce veľkosť, škálovacie faktory bloku a škálovací faktor domény. Tieto nastavenia sú striktné dodržiavané a ich nevhodné nastavenie môže spôsobiť nedosiahnutie cieľa (t.j. strata cieľovej kvality alebo prekročenie cieľovej veľkosti, v závislosti od nastaveného módu kompresie) alebo nájdenie nevhodného atraktora.

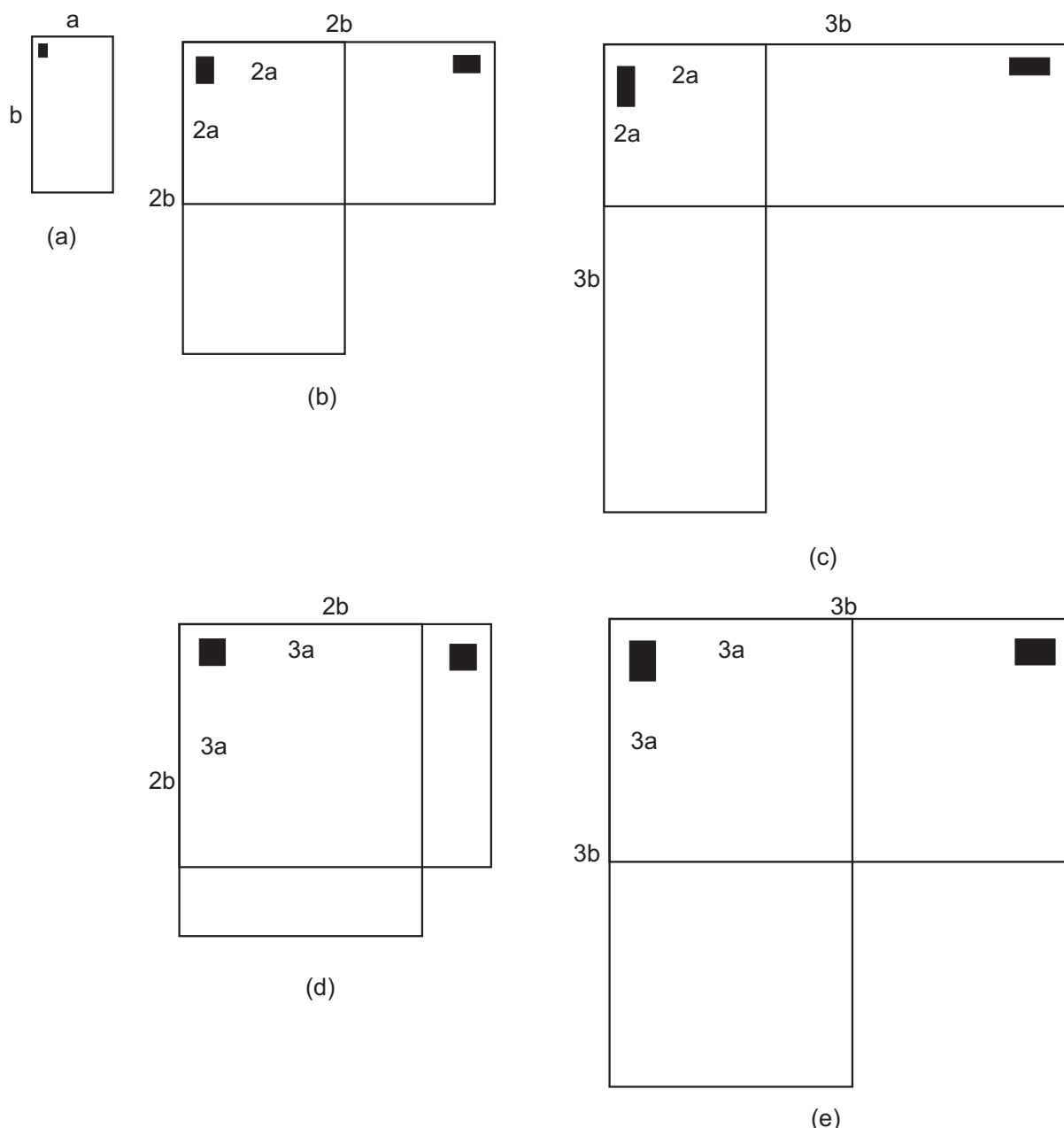
Minimálna povolená veľkosť bloku (*MinRangeSize*): blok, ktorý má obidva rozmery menšie alebo rovné ako nastavená hodnota, už nebude rozdelený na menšie bloky (aj keby to bolo potrebné). Hodnota *MinRangeSize=0* znamená neobmedzené delenie bloku (t.j. dovoľuje

algoritmu deliť blok až do rozmerov 1x1). Väčšie hodnoty zamedzujú generovaniu malých blokov, ale môžu spôsobiť stratu kvality.

Maximálna povolená veľkosť bloku (*MaxRangeSize*): blok, ktorý má aspoň jeden rozmer väčší ako nastavená hodnota, nebude pokrývaný a bude hneď rozdelený na menšie bloky (aj keby počet blokov presiahol povolenú hranicu). Hodnota *MaxRangeSize=0* povoľuje neobmedzenú veľkosť bloku. Menšie hodnoty zamedzujú pokrývaniu veľkých blokov (pre ktoré sa väčšinou nenájde vhodné pokrytie), čím sa ušetrí čas, ale môžu spôsobiť generovanie veľkého množstva blokov, čím sa zväčší veľkosť výstupu.

Minimálny pomer veľkosti bloku a domény (*MinRangeFactor*) a maximálny pomer veľkosti bloku a domény (*MaxRangeFactor*): tieto dva parametre ovplyvňujú výber domén. Na jednej pozícii sú vybrané domény so všetkými možnými kombináciami škálovacích faktorov, ktoré sú ešte kombinované s otočením (t.j. ak *MinRangeFactor=2* a *MaxRangeFactor=3* tak na jednej pozícii je vybraných a porovnaných 8 domén škálovaných nasledovne: 2x2, 2x2o, 2x3, 2x3o, 3x2, 3x2o, 3x3, 3x3o, vzhľadom na veľkosť bloku). Veľký rozsah faktorov spôsobí značné predĺženie pokrývania (čas potrebný na pokrytie rastie exponenciálne). Takisto veľké hodnoty faktorov spomaľujú pokrývanie.

Maximálny povolený škálovací faktor domény (*MaxScaleFactor*, mal by byť menší ako 1.0 ale nie je to podmienkou): pri pokrývaní bloku vybratou doménou sa vypočíta najlepší škálovací faktor a posunutie (v jasovej zložke obrazu, nie v priestorovej) pre dosiahnutie minimálnej RMS odchýlky. Ak ale škálovací faktor je väčší ako 1.0 výsledný IFS systém nemusí konvergovať resp. atraktor môže mať neobmedzené rozmery. Preto, ak vypočítaný škálovací faktor je väčší ako nastavená hodnota, pokrytie je odmietnuté.



Veľkosti domén pre  $MinRangeFactor=2$  a  $MaxRangeFactor=3$  (a) pôvodný blok s rozmermi  $a \times b$ . Kombinácie: (b)  $2 \times 2$  ( $2a \times 2b$ ) a  $2 \times 2o$  ( $2b \times 2a$ ), (c)  $2 \times 3$  ( $2a \times 3b$ ) a  $2 \times 3o$  ( $3b \times 2a$ ), (d)  $3 \times 2$  ( $3a \times 2b$ ) a  $3 \times 2o$  ( $2b \times 3a$ ), (e)  $3 \times 3$  ( $3a \times 3b$ ) a  $3 \times 3o$  ( $3b \times 3a$ ).

Nastavenia kvantizácie: hodnoty škálovacieho faktoru a posunutia vypočítané pri pokrývaní sa ďalej kvantujú lineárnym kvantizátorom a z kvantovaných hodnôt sa vypočíta RMS odchýlka. Väčší počet kvantizačných úrovní dovolí zachovať kvalitu, ale môže zväčšiť veľkosť výstupu. Menší počet úrovní znižuje kvalitu, ale znižuje veľkosť výstupu. V istých prípadoch menší počet úrovní môže spôsobiť generovanie väčšieho množstva blokov a tým zväčšiť výstup (hlavne pri vyšších kvalitách, kedy kvantovanie značne zníži kvalitu pokrytia bloku). Počet kvantizačných úrovní pre škálovací faktor domény určuje hodnota *ScaleLevels*. Počet kvantizačných úrovní pre posunutie určuje hodnota *OffsetLevels*.

Nastavenia výpočtových vlákien: trieda *HV encoder* je navrhnutá a implementovaná s možnosťou spolupráce viacerých výpočtových vlákien, ktoré zabezpečujú paralelné pokrývanie jedného bloku. Týmto dovoľuje využiť možnosti viac jadrových a viac

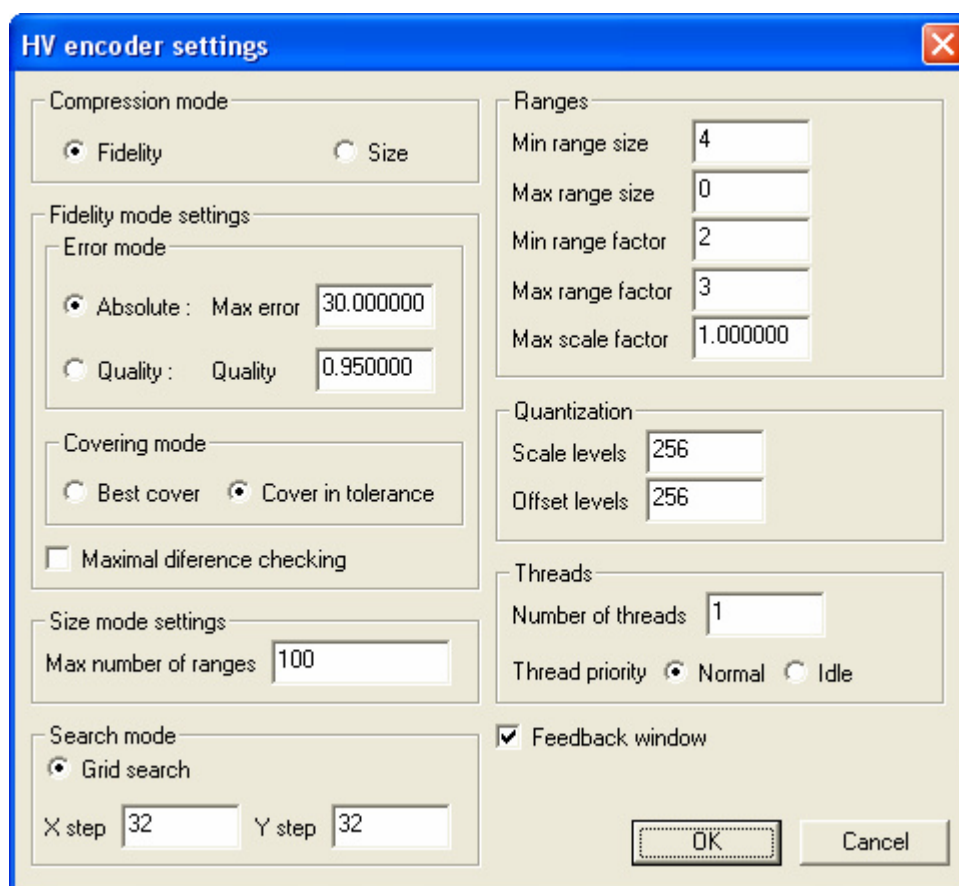
procesorových systémov a urýchliť proces pokrývania. Táto možnosť je v experimentálnom stave a nie je zaručená 100%-ná funkčnosť (odporúčame používať len jedno vlákno kvôli korektnosti výsledkov).

Počet paralelných spolupracujúcich vlákien určuje hodnota *NumThreads*.

Nastavenie systémovej priority výpočtových vlákien: voľba *TP\_NORMAL* nastaví normálnu prioritu, čo môže spôsobiť dlhšie nereagovanie systému kvôli vytáženiu procesora. Voľba *TP\_IDLE* nastaví prioritu na minimálnu hodnotu, čo zaručí reakcie systému, ale môže skresliť (predĺžiť) merané časy výpočtov.

Okno vizualizácie algoritmu: voľba *FeedbackWindow* povolí alebo zakáže zobrazenie vizualizačného okna, cez ktoré je možné monitorovať stav a priebeh algoritmu. Zapnutá vizualizácia môže výrazne spomaliť priebeh pokrývania.

## 5.2 Konfiguračný dialóg



Konfiguračný dialóg poskytuje voľby popísané v konfiguračnej štruktúre.

V bloku *Compression mode* je možné nastaviť mód kompresie. Voľba *Fidelity* nastaví mód *CM\_LOSSY\_FIDELITY* a voľba *Size* nastaví mód *CM\_LOSSY\_SIZE*. V bloku *Fidelity mode settings* sa nachádzajú nastavenia ovplyvňujúce mód *Fidelity* (napr. *Error mode*, *Covering mode*, *Maximal difference checking*). V tomto bloku je možné nastaviť aj cieľovú kvalitu výstupu, voľbou *Quality* a nastavením hodnoty *Quality*. V bloku *Size mode settings* sa nachádzajú nastavenia ovplyvňujúce mód *Size* (napr. *Max number of ranges*). V bloku *Search mode* je možnosť voľby prehľadávacieho algoritmu. Voľba *Grid search* zvolí mriežkové



prehľadávanie (*SM\_GRIDSEARCH*). Pod voľbou *Grid search* je možné nastaviť hustotu mriežky v x-ovom a v y-ovom smere. V bloku *Ranges* sa nachádzajú obmedzujúce nastavenia parametrov blokov (napr. *MinRangeSize*, *MaxRangeSize*, ... atď). V bloku *Quantization* je možné nastaviť počty kvantizačných úrovní pre kvantovanie škálovacích a posúvacích faktorov. V bloku *Threads* sú nastavenia výpočtových vlákien (*Number of threads*, *Thread priority*). Poslednou voľbou je voľba *FeedbackWindow*, ktorá povolí zobrazenie vizualizačného okna.

## 6. Definícia triedy

Trieda *HV encoder* je definovaná nasledovne:

```
class THVEncoderFilter : public TFilter {
public:
    HINSTANCE hInstance;           // identifikácia inštancie knižnice

    CThreadLock *ThreadLock;       // zámok na synchronizáciu vlákien

    TPin* CfgInPin;                 // vstupný konfiguračný pin
    TPin* CfgOutPin;                // výstupný konfiguračný pin
    TPin* InPin;                    // vstupný pin
    TPin* HDRPin;                   // výstupné piny...
    TPin* SIPin;
    TPin* OIPin;
    TPin* TRNPin;
    TPin* DPPin;
    TPin* DFPin;
    TPin* RPPin;
    TPin* RSPin;
    TPin* RTDPin;
    TPin* RTTPin;

    CPlane *InPlane;               // vstupná rovina
    CPlane *Sum;                   // rovina predpočítaných súm
    CPlane *Sum2;                  // rovina predpočítaných kvadratických súm

    CQuantizer *sQ;                // kvantizátor pre škálovacie koeficienty
    CQuantizer *oQ;                // kvantizátor pre posúvacie koeficienty

    CPointerStack *RangeStack;      // zásobník nespracovaných blokov
    CPointerArray *RangePool;       // pole pokrytých blokov
    CPointerArray *Threads;         // výpočtové vlákna
    CRange *ActiveRange;            // aktuálny blok
    CRange *Root;                   // strom blokov
    CDomainGenerator *DG;           // generátor domén

    CFeedbackWindow *FW;           // vizualizačné okno

    BOOL StopFlag;                  // indikátor zastavenia
    BOOL CorrectCover;              // indikátor korektnosti pokrytia
}
```

```

int RangeCounter;           // počítadlo blokov
DET_FLOAT progress;        // kompletnosť operácie

int CompressionMode;        // konfigurácia...
float MaxError;
BOOL MaxDifCheck;
int MaxNumRanges;
int MinRangeSize;
int MaxRangeSize;
int MinRangeFactor;
int MaxRangeFactor;
float MaxScaleFactor;
int OffsetQLevels;
int ScaleQLevels;
int SearchMode;
int GridSearch_StepX;
int GridSearch_StepY;
int CoveringMode;
int ErrorMode;
float Quality;
int NumThreads;
int ThreadPriority;
BOOL FeedbackWindow;

DET_FLOAT smax;             // konštanty...
DET_FLOAT smin;
DET_FLOAT omax;
DET_FLOAT omin;
DET_FLOAT inmax;
DET_FLOAT inmin;
DET_FLOAT emax;

THeader Header;            // výstupné štruktúry...
TIntArray ScaleIndexies;
TIntArray OffsetIndexies;
TIntArray Transformations;
TIntArray DomainPositions;
TIntArray DomainFactors;
TIntArray RangePositions;
TIntArray RangeSizes;
TIntArray RangeTreeDistanties;
TIntArray RangeTreeTypes;

int calcConstants();        // pomocné funkcie...
int createSumPlanes();
int createThreads();
int createQuantizers();
int freeTempResources();
int makeOutputData();

```

```

int getNextDomain(CDomain *dest, CRange **range);
BOOL checkDomain(CDomain *domain);
int Result(DET_FLOAT error, DET_FLOAT maxdif, CDomain *domain, CRange *range);
int buildRange(CRange *range);
int generateId();
int updateRange();
int calcPartition(CRange *range, int &dist, int &type);
int makePartition(CRange *range, int dist, int type);
CRange* findRangeWithMaxError();
CRange* getRangeToPartition(int &dist, int &type);

int setConfigFromPin(TPin *pin);           // funkcie pre konfigurovanie z pinov...
int putConfigOnPin(TPin *pin);
int freeConfigOnPin(TPin *pin);
int clearOutputPins();                     // funkcie pre čistenie výstupných pinov...
int freeOutputData();

public:
    THVEncoderFilter();                     // konštruktor
    THVEncoderFilter(HINSTANCE hInst);      // konštruktor s parametrami
    ~THVEncoderFilter();                     // deštruktor

int setConfigData(TBuffer config, int type); // predefinované funkcie triedy TFilter...
TBuffer getConfigData(int type);
int initialize();
int run();
int reset();
int stop();
int finalize();
int showConfigDialog();
};

```

## 7. Príklad použitia

