Rensselaer

CNGV

Embedded Set Partition Coding

William A. Pearlman

Professor and Director,

Center for Next Generation Video

Rensselaer Polytechnic Institute

http://www.rpi.edu/web/NGV

http://www.cipr.rpi.edu/staff/pearlman.html

pearlman@rpi.edu

Rensselaer

CNGV

Outline

- SPECK (Set Partition Embedded bloCK) coding
- Features
  - Low complexity
  - Embeddedness
  - Implementation issues
- Results and Images
- Conclusions

Rensselaer

CNGV

Low Complexity Set Partitioning Embedded Coder: SPECK

- SPECK discovered by Islam and Pearlman (VCIP'99, Proc. SPIE 3653) and possibly others
  - Partitions and codes sets of coefficients grouped within subbands
- Can encode any grouping of pixels or coefficients
- Bitstream is *embedded* – lower rate image can be decoded from subset of bitstream

Rensselaer

CNGV

Partitioning of Significant Set *S* into Offspring

Operate through significance decisions for partitioning sets of {wavelet transform} coefficients.
Sets and partitioning rules differ between SPECK and SPIHT.

Significance test for pixels $c_{i,j}$ in set S

$$S_n(S) = \begin{cases} 1, & \text{if } \left(\max_{(i,j)\in S} \left|c_{ij}\right|\right) \geq 2^n, \\ 0, & \text{otherwise.} \end{cases}$$

$O_0(S)$	$O_1(S)$
$O_2(S)$	$O_3(S)$

Rensselaer

CNGV

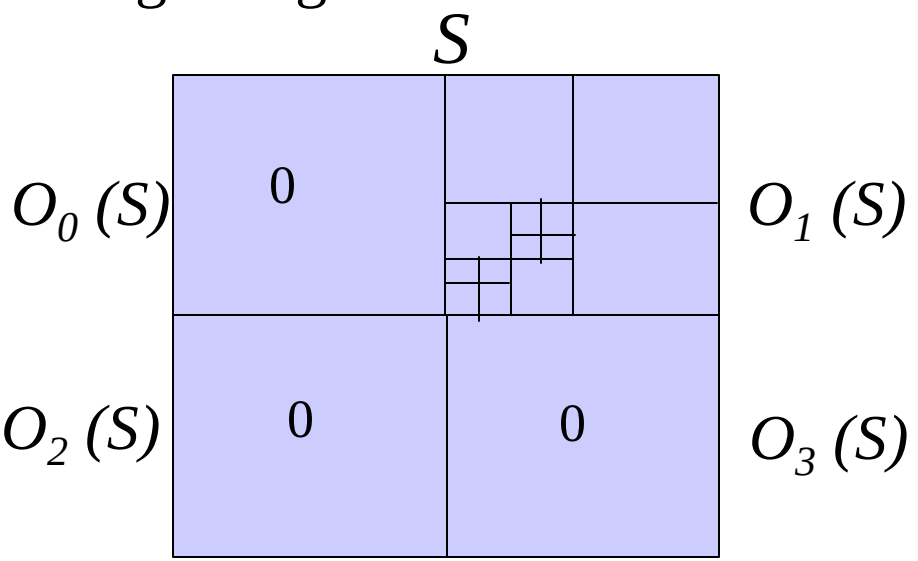
Set Partitioning Rules of SPECK

S Partitioned into Offspring Sets

Recursive quadrature splitting of significant sets

LIS - list of insignificant sets in increasing order of size starting with single element sets

LSP - list of significant single elements.

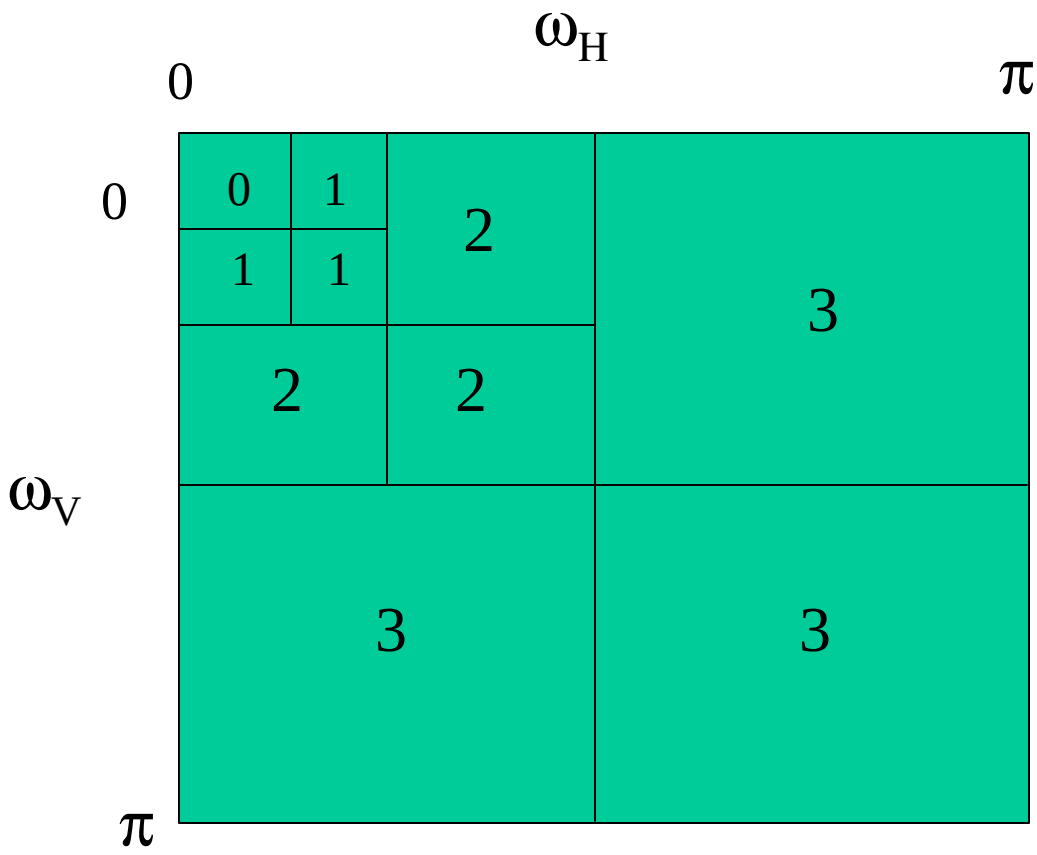


For given threshold or bit plane *n*, insignificant sets and coefficients go to LIS, significant coefficients to LSP. Decrease n by 1, repeat.

Rensselaer

CNGV

Multi-resolution (Wavelet) Transform



0 -> 1/8 res.
+
1,1,1 -> 1/4 res.
+
2,2,2 -> 1/2 res.
+
3,3,3 -> full res.

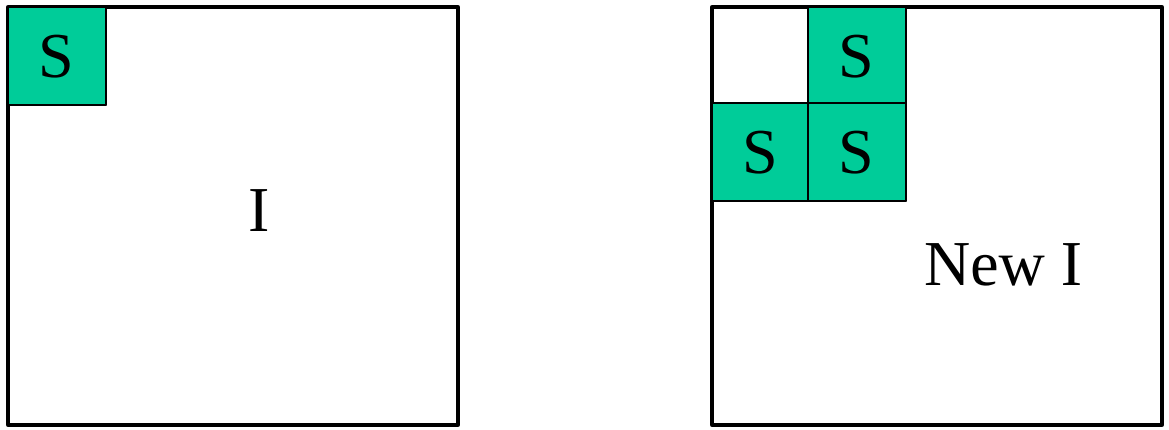
Rensselaer

CNGV

Octave Band Partitioning

Test through full transform for each n,

$$n = n_{\max}, n_{\max} - 1, n_{\max} - 2, \ldots \quad n_{.\max} = \max_{i,j} \lfloor \log_2 | c_{i,j} | \rfloor$$



S is coded

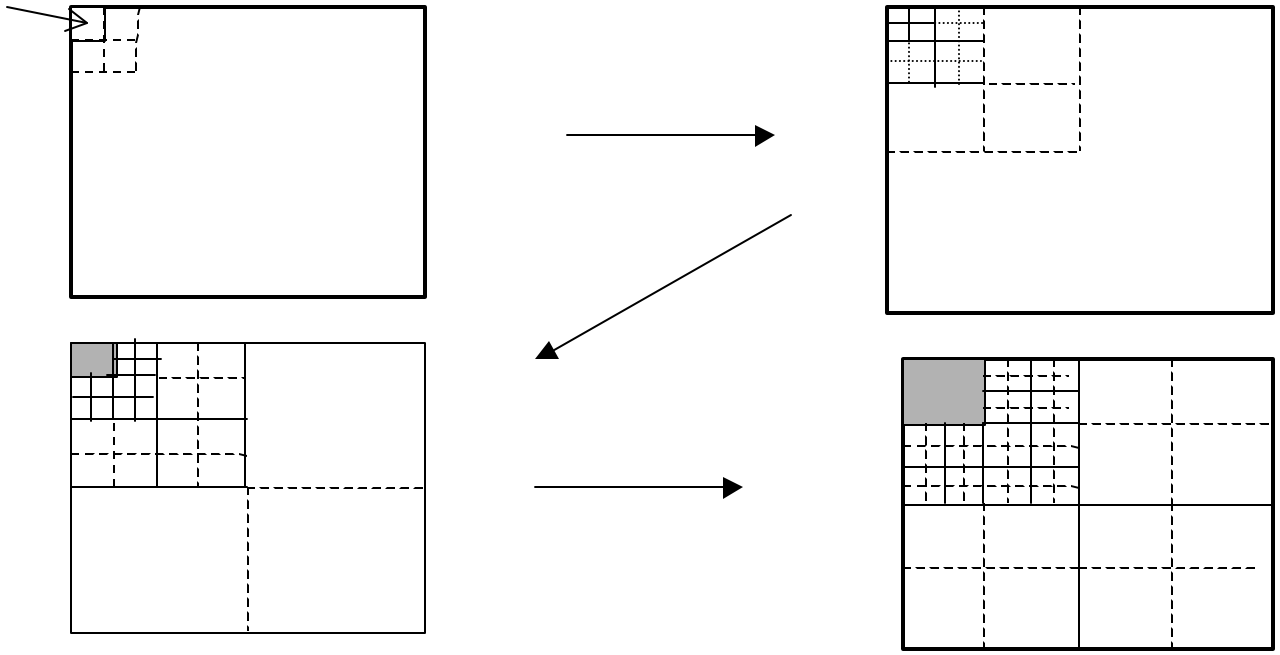
Rensselaer

CNGV

Subband Partitioning Order

- Partitioning proceeds from coarse to fine scale

Initial S



Rensselaer

Coding Sequence

CNGV

- SPECK uses two lists LIS and LSP visited for significance testing in that order, for a given bit-plane (*n*).
  - LIS: list of coordinates of insignificant coefficients organized into sublists of increasing size from single element sets at the top
    - Functionally equivalent to separate LIP list
  - LSP: list of coordinates of significant pixels
    - Send n-th bit of all pixels found to be significant in previous passes.

Rensselaer

CNGV

LSP Sorting by Magnitude and Progressive Bit-Plane Transmission

Transmission of magnitude-sorted coefficients

	sign	s	s	s	s	s	s	s	s	s	s	s	s	s
msb	5	1	1	0	0	0	0	0	0	0	0	0	0	0
	4	®	®	1	1	0	0	0	0	0	0	0	0	0
	3	®	®	®	®	1	1	1	1	0	0	0	0	0
	2	®	®	®	®	®	®	®	®	1	1	1	1	1
	1	®	®	®	®	®	®	®	®	®	®	®	®	®
lsb	0	®	®	®	®	®	®	®	®	®	®	®	®	®

Send *n*-th bit of all coefficients found to be significant in previous passes.

Rensselaer

CNGV

Transmission of Bits

- LSP sign and refinement bits sent raw
  - Possible to encode by context-based arithmetic coding to achieve small gains
- Significance decision encoded or sent raw
  - Context-based arithmetic coding in 2x2 sets achieves some gains
  - Most entropy-coding gain can be obtained with fixed or semi-adaptive Huffman codes
- Fully adaptive entropy-coded version of SPECK called EZBC surpasses JPEG2000

Pseudo-Code of SPECK Algorithm

1. **Initialization**

   - Partition image transform $\mathcal{X}$ into two sets: $\mathcal{S} \equiv$ root, and $\mathcal{I} \equiv \mathcal{X} - \mathcal{S}$
   - output $n = \lfloor \log_2(\max_{\forall (i,j) \in \mathcal{X}} |c_{i,j}|) \rfloor$
   - add $\mathcal{S}$ to LIS and set LSP $= \phi$

2. **Sorting Pass**

   - in increasing order of size $\mathcal{C}$ of sets

- for each set $\mathcal{S} \in$ LIS,
  - `ProcessS(`$\mathcal{S}$`)`

- `ProcessI()`

3. **Refinement Pass**

   - for each $(i, j) \in$ LSP, except those included in the last sorting pass, output the nth MSB of $|c_{i,j}|$

4. **Quantization Step**

   - decrement n by 1, and go to step 2

`ProcessS`($\mathcal{S}$) $\{$

- output $\mathcal{S}_n(\mathcal{S})$
- if $\mathcal{S}_n(\mathcal{S}) = 1$
  - if $\mathcal{S}$ is a coefficient, output sign of $\mathcal{S}$ and add $\mathcal{S}$ to LSP
  - else `CodeS`($\mathcal{S}$)
  - if $\mathcal{S} \in$ LIS, remove $\mathcal{S}$ from LIS
- else

- if $\mathcal{S} \notin$ LIS, add $\mathcal{S}$ to LIS }

`CodeS`($\mathcal{S}$) {

- Partition $\mathcal{S}$ into four equal subsets $\mathcal{O}(\mathcal{S})$
- for each $\mathcal{O}(\mathcal{S})$
  - output $\mathcal{S}_n(\mathcal{O}(\mathcal{S}))$
  - if $\mathcal{S}_n(\mathcal{O}(\mathcal{S})) = 1$
    - if $\mathcal{O}(\mathcal{S})$ is a coefficient, output sign of $\mathcal{O}(\mathcal{S})$ and add $\mathcal{O}(\mathcal{S})$ to LSP

        * else `CodeS`($\mathcal{O}(\mathcal{S})$)

    - else

        * add $\mathcal{O}(\mathcal{S})$ to LIS }

`ProcessI()` {

- output $\mathcal{S}_n(\mathcal{I})$

- if $\mathcal{S}_n(\mathcal{I}) = 1$

    - `CodeI()` }

`CodeI()` {

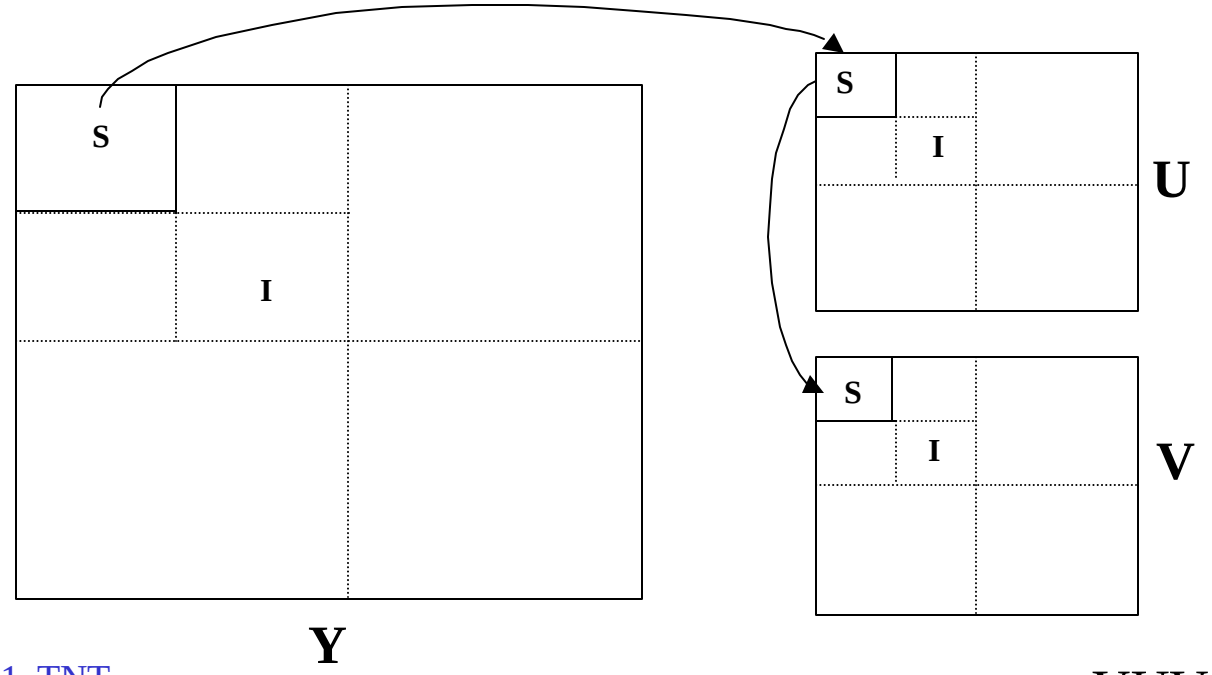
- Partition $\mathcal{I}$ into four sets — three $\mathcal{S}$ and one $\mathcal{I}$
- for each of the three sets $\mathcal{S}$
  - `ProcessS(`$\mathcal{S}$`)`
- `ProcessI()` }

Comment	Point or Set Tested	Output Bit	Action	Control Lists
n=5 Sorting				LIS = {(0,0)**1**} LSP = ϕ
	$S^1(0,0)$	1	quad split, add to LIS(0)	LIS ={ (0,0)**0**, (0,1)**0**,(0,1)**0**,(1,0)**0**,(1,1)**0**} LSP = ϕ
	(0,0)	1+	(0,0) to LSP	LIS = {(0,1)**0**,(0,1)**0**,(1,0)**0**,(1,1)**0**} LSP = {(0,0)}
	(0,1)	1-	(0,1) to LSP	LIS = {(1,0)**0**,(1,1)**0**} LSP = {(0,0),(0,1)}
	(1,0)	0	none	
	(1,1)	0	none	
	$S^1(0,2)$	1	quad split, add to LIS(0)	LIS = {(1,0)**0**,(1,1)**0**},(0,2)**0**,(0,3)**0**}(1,2)**0**,(1,3)**0**}
	(0,2)	1+	(0,2) to LSP	LSP = {(0,0),(0,1),(0,2)} LIS = {(1,0)**0**,(1,1)**0**},(0,3)**0**}(1,2)**0**,(1,3)**0**}
	(0,3)	0	none	
	(1,2)	0	none	
	(1,3)	0	none	
	$S^1(2,0)$	0	add to LIS(1)	LIS = {(1,0)**0**,(1,1)**0**},(0,3)**0**}(1,2)**0**,(1,3)**0**,(2,0)**1**}
	$S^1(2,3)$	0	add to LIS(1)	LIS = {(1,0)**0**,(1,1)**0**},(0,3)**0**}(1,2)**0**,(1,3)**0**,(2,0)**1**,(2,3)**1**}
	$S^2(0,4)$	0	add to LIS(1)	LIS = {(1,0)**0**,(1,1)**0**},(0,3)**0**}(1,2)**0**,(1,3)**0**,(2,0)**1**,(2,3)**1**,(0,4)**2**}
	$S^2(4,0)$	1	quad split, add to LIS(1)	LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**,(1,2)**0**,(1,3)**0**,(2,0)**1**,(2,3)**1**,(4,0)**1**,(4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**}
	$S^1(4,0)$	0	none	
	$S^1(4,2)$	1	quad split, add to LIS(0)	LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**,(1,2)**0**,(1,3)**0**,(4,2)**0**,(4,3)**0**,(5,2)**0**,(5,3)**0**,(2,0)**1**,(2,3)**1**, (4,0)**1**,(4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**}
	(4,2)	0	none	
	(4,3)	1+	move (4,3) to LSP	LSP = {(0,0),(0,1),(0,2),(4,3)} LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**}(1,2)**0**,(1,3)**0**,(4,2)**0**,(5,2)**0**,(5,3)**0**,(2,0)**1**,(2,3)**1**,(4,0)**1**, (4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**}
	(5,2)	0	none	
	(5,3)	0	none	
	$S^1(6,0)$	0	none	
	$S^1(6,2)$	0	none	
End n=5 Sorting	$S^2(4,4)$	0	add to LIS(2)	LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**,(1,2)**0**,(1,3)**0**,(4,2)**0**,(5,2)**0**,(5,3)**0**,(2,0)**1**,(2,3)**1**,(4,0)**1**, (4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**, (0,4)**2**} LSP = {(0,0),(0,1),(0,2),(4,3)}

Comment	Point or Set Tested	Output Bit	Action	Control Lists
n=4 Sorting				LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**,(1,2)**0**,(1,3)**0**,(4,2)**0**,(5,2)**0**,(5,3)**0**,(2,0)**1**,(2,3)**1**,(4,0)**1**, (4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**, (0,4)**2**} LSP = {(0,0),(0,1),(0,2),(4,3)}
Test LIS(0)	(1,0)	1-	(1,0) to LSP	
	(1,1)	1+	(1,1) to LSP	LIS = {(0,3)**0**,(1,2)**0**,(1,3)**0**,(4,2)**0**,(5,2)**0**,(5,3)**0**,(2,0)**1**,(2,3)**1**,(4,0)**1**, (4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**, (0,4)**2**} LSP = {(0,0),(0,1),(0,2),(4,3),(1,0),(1,1)}
	(0,3)	0	none	
	(1,2)	0	none	
	(1,3)	0	none	
	(4,2)	0	none	
	(5,2)	0	none	
	(5,3)	0	none	
Test LIS(1)	$S^1(2,0)$	0	none	
	$S^1(2,3)$	0	none	
	$S^1(4,0)$	0	none	
	$S^1(6,0)$	0	none	
	$S^1(6,2)$	0	none	
Test LIS(2)	$S^2(0,4)$	0	none	
	$S^2(4,4)$	0	none	
Refinement	(0,0)	1	decoder adds 2^4	
	(0,1)	0	decoder subtracts 0	
	((0,2)	1	decoder adds 2^4	
	(4,3)	0	decoder adds 0	
End n=4				

Color SPECK Coding

- For a given threshold; the luminance component is first divided into an ‘S’ and an ‘I’ set (see figure) and processed.
- Following this, the chrominance components are divided into an ‘S’ and an ‘I’ set and processed ***at the same threshold***.
- The threshold is reduced by half (bit-plane coding) after the refinement pass and the algorithm repeats.



YUV 4:2:0 assumed

Rensselaer

CNGV

Comparison of various color codecs

KBits	Y/C	JPEG 2000	SPIHT	CSPECK	MPEG-4
20.9	Y	34.65	35.40	**35.83**	35.09
(Hall)	C	**41.27**	39.73	37.62	39.78
28.5	Y	37.73	38.05	**38.50**	37.82
(Hall)	C	**43.14**	40.71	39.79	41.16
19.3	Y	33.65	34.33	**34.77**	33.95
(foreman)	C	**42.54**	40.32	37.42	40.56
28.7	Y	37.42	37.53	**37.71**	37.11
(foreman)	C	**43.70**	43.04	40.61	42.08

these figures indicate PSNR (in dB)

Original Image “Foreman.QCIF”

CSPECK (19.3 Kbits)

SPIHT (19.3 Kbits)

JPEG2000 (19.3 Kbits)

Rensselaer

CNGV

Motion-SPECK vs. Motion-JPEG2000

Codec	Mean PSNR in dB			Decoding Times * (sec)
	Y	U	V	
"Carphone"(97 frames) at 0.5 bits/pixel/frame				
Motion-JPEG2000	31.23	38.28	39.17	8.43
Motion-SPECK	32.56	35.97	36.25	8.74
"Trevor"(150 frames) at 0.25 bits/pixel/frame				
Motion-JPEG2000	28.59	36.73	37.28	11.39
Motion-SPECK	30.59	34.26	35.14	12.10

*Decoding time is based on Sun SPARC 10

(a)

(b)

(c)

(a) Original image, frame no. 77 of “Carphone” sequence

(b) Motion-SPECK reconstructed (0.5bpp/frame)

(c) Motion-JPEG2000 reconstructed (0.5bpp/frame)

Rensselaer

CNGV

Embedding Issues

- Full transform SPECK (and SPIHT) fully bit-embedded
  - next bit conveys less value information than previous one
- Full bit-embedding costs in speed and complexity
  - many applications do not require full bit-embedding and/or can not afford it (viz., geographic images, real-time animation)
- SPECK (and SPIHT) allow relaxation to *value embedding* by skipping refinement pass and sending all lower bit planes immediately for each new LSP coefficient.
  - *can not be done in JPEG2000*

Rensselaer

CNGV

Small Memory Implementations

- SPECK used in small subband blocks (SBHP)
  - replaced coding of subband blocks in JPEG2000 framework
- SPIHT implemented in JPEG2000 framework
  - SPIHT coding applied to small transform tree-blocks independently
    - Same could be done with SPECK
- Embedded bit streams from independent tree-block or subband block encoders are packetized and multiplexed to generate a progressive main bit stream

Rensselaer

Conclusions

CNGV

- Wavelet transform and set partition coding bring many desirable features:
  - Scalability in resolution and fidelity
  - Low complexity in computation and memory
    - Fast encoding and decoding
- Key to low complexity
  - set partitioning and testing for set significance
    - Adaptive techniques with higher complexity
      - SLCC (Chai et al), PACC (Marpe & Cycon), GTW (Hong & Ladner)
- Other features
  - Region of interest (ROI) fidelity enhancement
  - Error resilient bitstream
  - Parallelizable for embedded hardware applications