An Improved Progressive Lossy-to-Lossless Coding Method for Arbitrarily-Sampled Image Data

Michael D. Adams

Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC, V8W 3P6, Canada

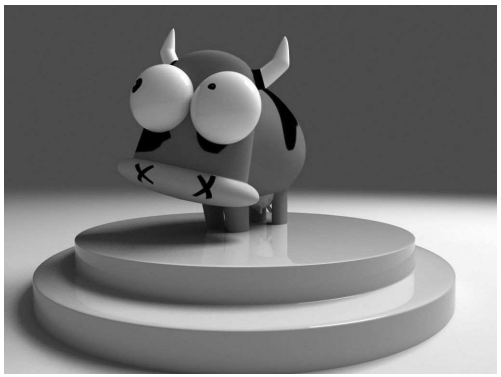
E-mail: mdadams@ece.uvic.ca

August 2013

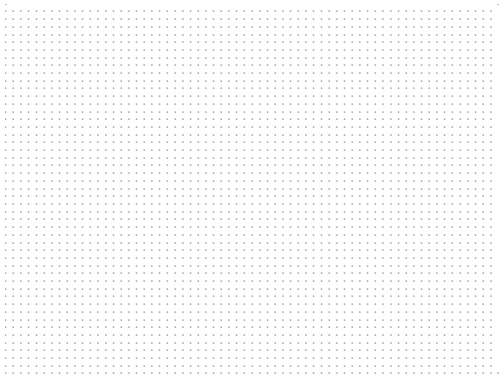
Outline

Motivation

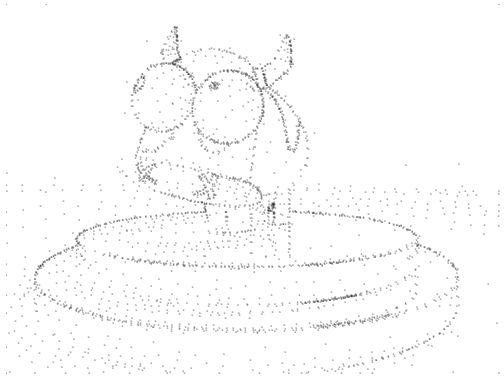
- since most images nonstationary, uniform sampling on lattice not optimal
- in regions where image has more detail, sampling density may be too low
- in regions where image has less detail, sampling density probably too high
- motivates arbitrary (i.e., nonuniform) sampling
- when arbitrary sampling employed, need means to efficiently encode resulting datasets
- example: arbitrarily-sampled datasets arise when triangle meshes (e.g., interpolants over Delaunay triangulations) used for image representation



Image

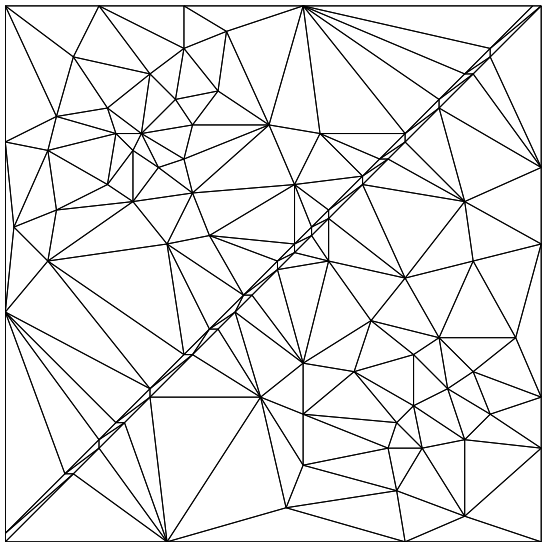


Uniform Sampling

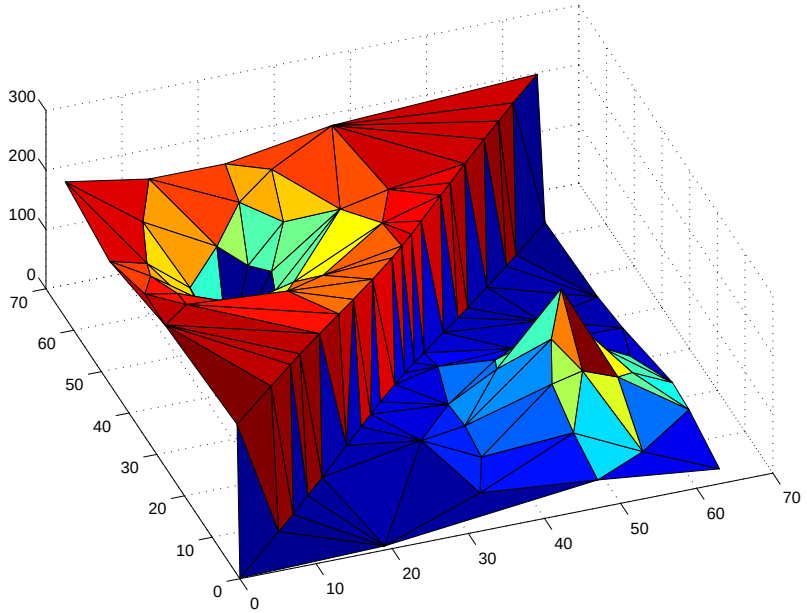


Arbitrary Sampling

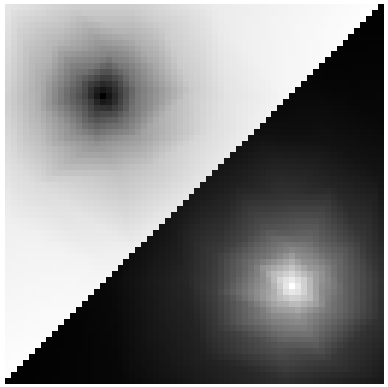
Mesh Model of Image (Sampling Density 2.5%)



Delaunay Triangulation of Image Domain



Resulting Triangle Mesh



Reconstructed Image

Arbitrarily-Sampled Image Dataset

- image is integer-valued function f defined for integer points (x, y) in the domain $D = \{0, 1, \ldots, W-1\} \times \{0, 1, \ldots, H-1\}$
- without loss of generality, assume that range of f can be represented as P-bit unsigned integers
- arbitrarily-sampled image dataset consists of:
  1. set $S = \{(x_i, y_i)\}_{i=0}^{|S|-1}$ of sample points, with $S \subset \mathbb{Z}^2$
  2. corresponding set $Z = \{z_i\}_{i=0}^{|S|-1}$ of sample values, where $z_i = f(x_i, y_i)$
- sampling density of dataset defined as $|S| / |D|$
- want to be able to efficiently code this type of dataset
- want progressive lossy-to-lossless coding capability

Average-Difference (AD) Transform

- average-difference (AD) transform maps two integers x_0 and x_1 to two integers y_0 and y_1, as given by

$$y_0 = \left\lfloor \tfrac{1}{2}(x_0 + x_1) \right\rfloor \quad \text{and} \quad y_1 = x_1 - x_0$$

- y_0 and y_1 correspond to approximate average of x_0 and x_1 and difference between x_0 and x_1, respectively
- if x_0 and x_1 can each be represented with n bits, y_0 and y_1 can be represented with n and $n+1$ bits, respectively
- inverse of above transform given by

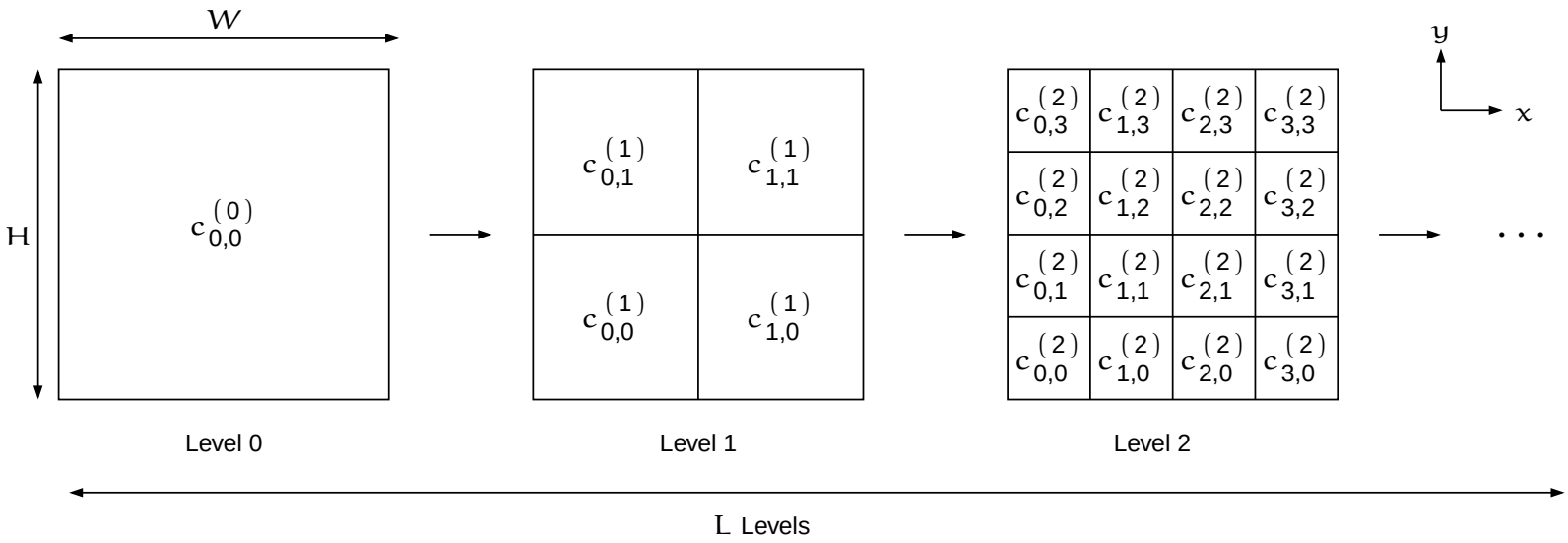
$$x_0 = y_0 - \left\lfloor \tfrac{1}{2} y_1 \right\rfloor \quad \text{and} \quad x_1 = y_1 + x_0$$

Average-Difference Image Tree (ADIT)

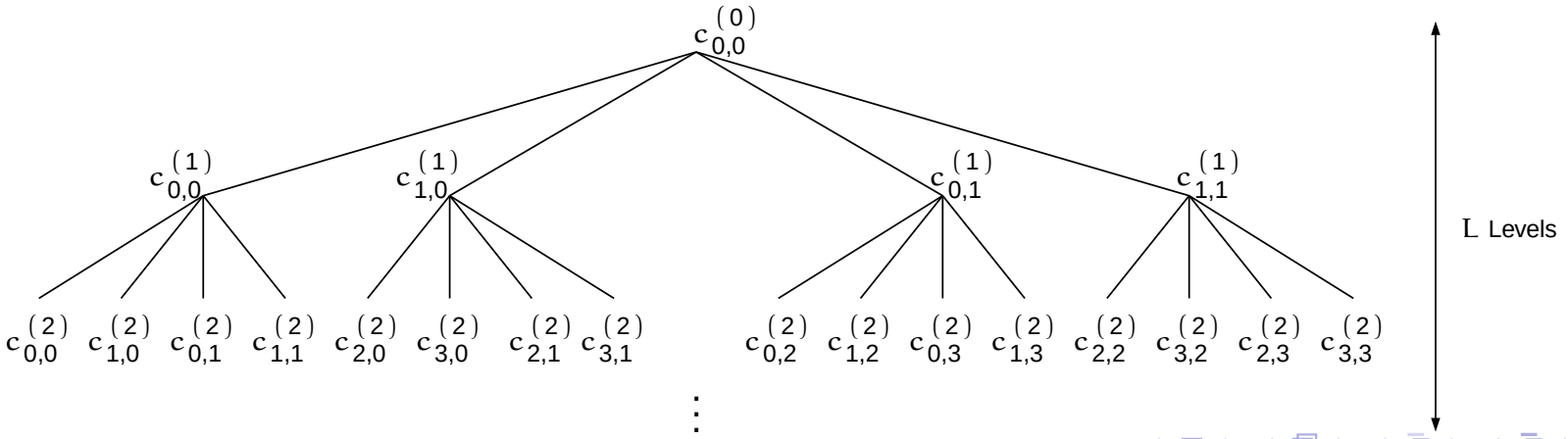
- average-difference image tree (ADIT) is tree-based data structure
- associated with quadtree partitioning of image domain, which splits image domain into cells
- each node in tree has zero to four children and is associated with following information:
  1. rectangular region in image domain, called cell
  2. one approximation coefficient
  3. $\max\{c-1, 0\}$ detail coefficients, where c is number of children possessed by node
- approximation coefficient: corresponds to approximate average value of all sample values contained in node's cell
- detail coefficients: specify difference between approximation coefficient of node and approximation coefficients of children

Quadtree Partitioning of Image Domain

- L-level quadtree partitioning of image domain D (where $L = \lceil \log_2 \max\{W, H\} \rceil + 1$):



- quadtree partitioning shown explicitly as tree:



ADIT Example

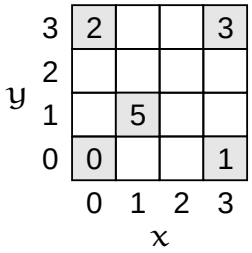
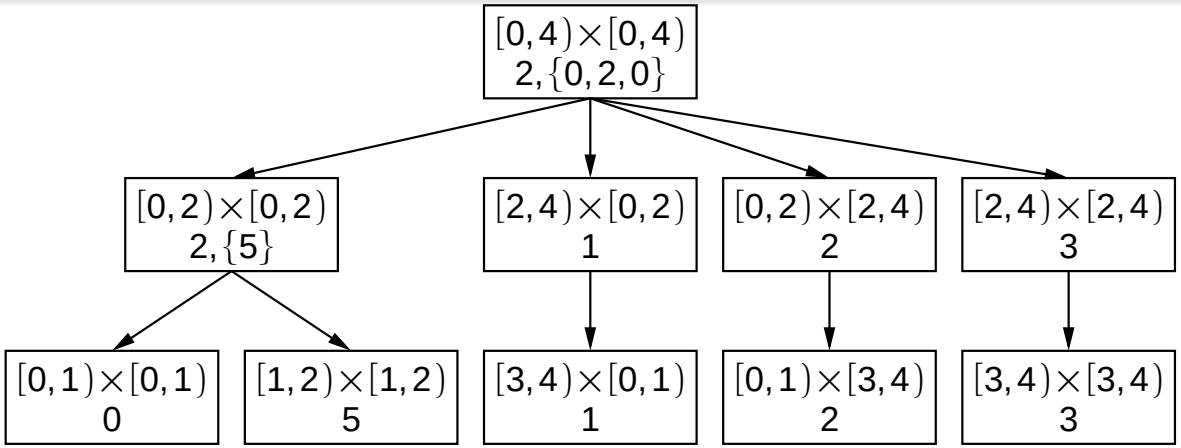


Image Dataset

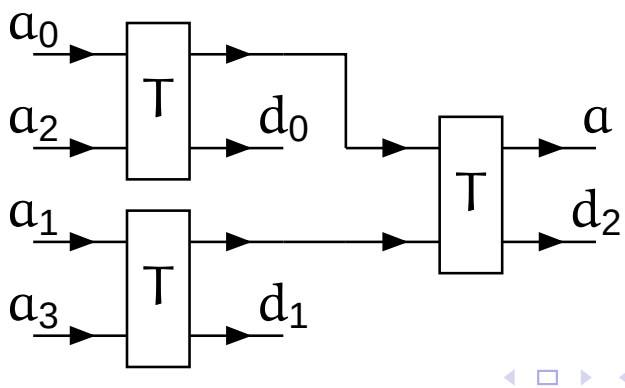


Corresponding ADIT

- each node in ADIT labelled (in order) with
  1. associated cell
  2. approximation coefficient
  3. detail coefficients if any, appearing in brace brackets
- image dataset can be losslessly reconstructed from information in leaf nodes
- can generate approximations of dataset from pruned versions of tree
- generate one sample point and corresponding sample value for each leaf node in pruned tree
- if leaf node has cell with area greater than one, corresponding sample point taken to be (approximate) centroid of cell

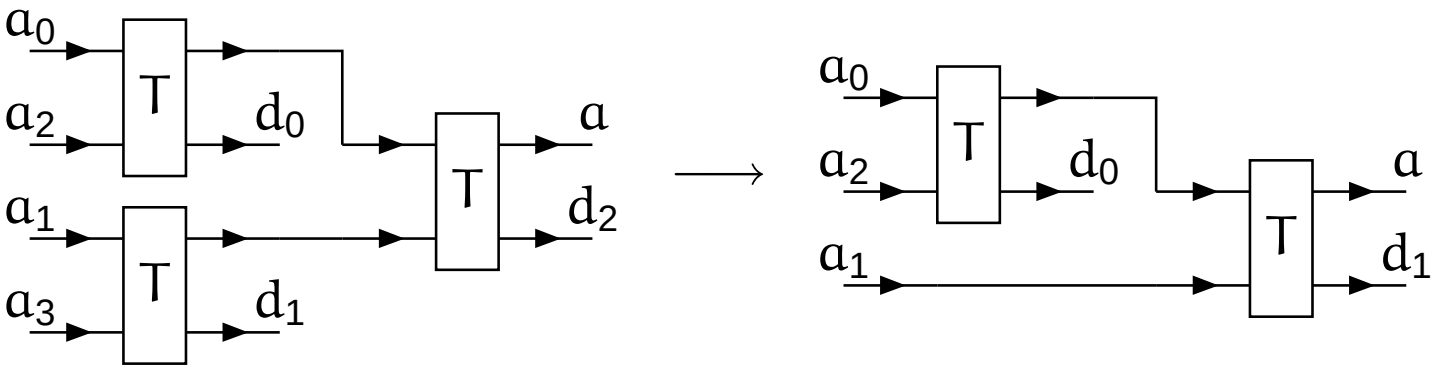
Approximation and Detail Coefficients

- consider arbitrary node n with c children, which has one approximation coefficient a and m detail coefficients $\{d_i\}_{i=0}^{m-1}$, where $m = \max\{c-1, 0\}$.
- for terminal node (which corresponds to sample point):
  - a chosen as sample value of corresponding sample point
  - no detail coefficients (since $c = 0$)
- for nonterminal node n:
  - a and $\{d_i\}_{i=0}^{m-1}$ computed through repeated application of AD transform
- let $\{a_i\}_{i=0}^{3}$ denote approximation coefficients of four possible children of node n
- if all four possible children present (i.e., $c = 4$), then $m = 3$, and a and $\{d_i\}_{i=0}^{m-1}$ are computed by applying AD transform repeatedly as shown in figure:



Approximation and Detail Coefficients (Continued)

- if some of the four possible children not present (i.e., $c < 4$), then in effect one or more of $\{a_i\}_{i=0}^{3}$ missing
- to handle missing inputs, apply following two rules to block diagram for case of no missing inputs (on previous slide):
  1. any transform block with only one input replaced by identity block that does not produce any detail coefficient
  2. if any transform block has no inputs, block is simply removed altogether
- for example, if $c = 3$ and child corresponding to a_3 not present, $m = 2$, and a and $\{d_i\}_{i=0}^{m-1}$ calculated as shown in right figure:



- each approximation coefficient representable as P-bit unsigned integer
- each detail coefficient representable as $(P+1)$-bit signed integer

ADIT Coding

- provides progressive lossy-to-lossless coding capability
- consider only encoding process (as decoding mirrors encoding)
- to (nonredundantly) capture all information in ADIT, sufficient to encode:
  1. width W and height H of image domain
  2. child configuration (CC) of each node (i.e., which of four possible children are present for each node)
  3. approximation coefficient a_r of root node
  4. detail coefficients (DCs) (if any) of each node
- aside from ADIT, two key data structures employed by encoder:
  1. child configuration (CC) queue
  2. detail coefficients (DC) queue
- CC queue:
  - priority queue
  - holds nodes whose CC information not yet coded
  - node priorities correspond to breadth-first traversal order
- DC queue
  - first-in first-out (FIFO) queue
  - holds nodes whose CC information has been coded but whose DC information not yet fully coded

ADIT Encoding Algorithm

- image width W, image height H, approximation coefficient a_r of root node, P bits/sample
- output header containing W, H, a_r, and P
- initialize context-adaptive binary arithmetic-coding engine, used to encode remainder of data
- clear CC and DC queues
- place root node on CC queue
- alternate between coding CC information for nodes on CC queue and DC information for nodes on DC queue
- switch queues when queue currently being processed becomes empty or maximum entropy budget exhausted
- after node on CC queue processed, node moved to DC queue and each of its children placed on CC queue
- after node on DC queue processed, node placed on DC queue only if still has more DC information remaining to be coded
- terminates when both queues empty

ADIT Encoding Algorithm (Detailed)

1: ccBudget := 512
2: dcBudget := 256
3: encode header information (i.e., W, H, P, and a_r)
4: insert root node on CC queue
5: **while** CC queue not empty or DC queue not empty **do**
6: **while** ccBudget > 0 and CC queue not empty **do**
7: set node to element at front of CC queue and remove element from queue
8: encode CC information for node and set b to entropy of information just coded
9: ccBudget := ccBudget - b
10: insert each child of node on CC queue
11: insert node on DC queue
12: **end while**
13: **while** dcBudget > 0 and DC queue not empty **do**
14: set node to element at front of DC queue and remove element from queue
15: invoke DC encoding process for node and set b to entropy of information just coded
16: **if** still more DC data to encode for node **then**
17: insert node on DC queue
18: **end if**
19: dcBudget := dcBudget - b
20: **end while**
21: ccBudget := min{512, ccBudget + 512}
22: dcBudget := min{256, dcBudget + 256}
23: **end while**

CC and DC Encoding

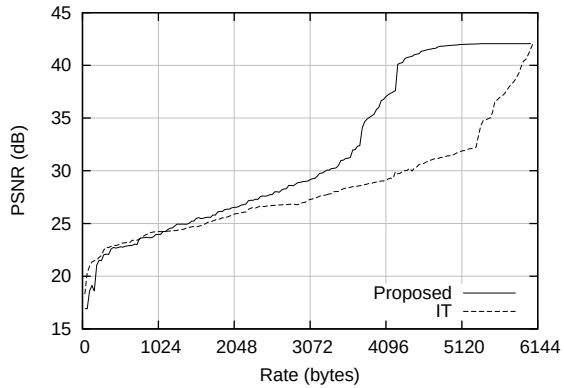
- CC encoding performed same as in IT method
- each detail coefficient is $(\mathrm{P}+1)$-bit signed integer (i.e., P-bit magnitude plus sign bit)
- each detail coefficient for given node is coded using $\mathrm{SI}(\mathrm{P}, \min\{\mathrm{P}, 4\})$ binarization, conditioned on node's level in tree
- each time DC encoding process invoked for particular node, one more magnitude bit is coded (starting from most-significant bit position) for each of node's detail coefficients
- whenever first nonzero magnitude bit coded for detail coefficient, it is immediately followed by sign bit
- all of DC information for node will be encoded after coding process has been invoked P times for node (since each invocation of DC coding process codes one magnitude bit from each of node's detail coefficients and each detail coefficient has P magnitude bits)

Lossless Coding Performance

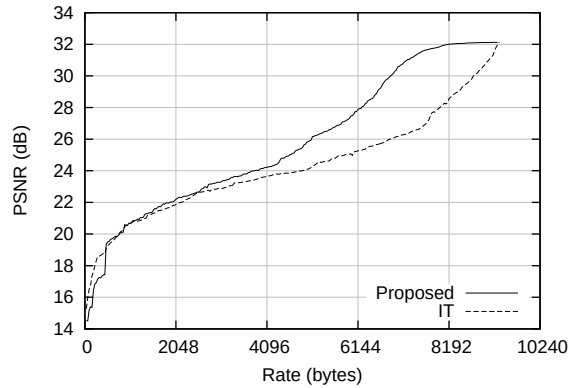
Image	Sampling Density (%)	Rate (Bytes)	
		Proposed	IT
ct	0.5	**3266**	3279
	1.0	**6054**	6086
	2.0	**11077**	11170
	4.0	**20106**	20300
lena	0.5	**2779**	2797
	1.0	**5098**	5115
	2.0	**9284**	9329
	4.0	**16729**	16846
peppers	0.5	**2827**	2833
	1.0	**5217**	5229
	2.0	**9527**	9571
	4.0	**17180**	17285

- proposed method consistently yields slightly lower lossless bit rates
- greatly improved progressive coding performance at lossy rates does not come at cost of reduced lossless coding efficiency

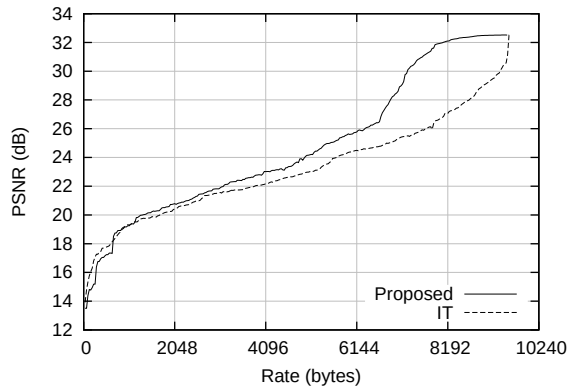
Progressive Coding Performance



`ct` at sampling density of 1%
(mean and maximum difference 3.45 dB and 10.88 dB)



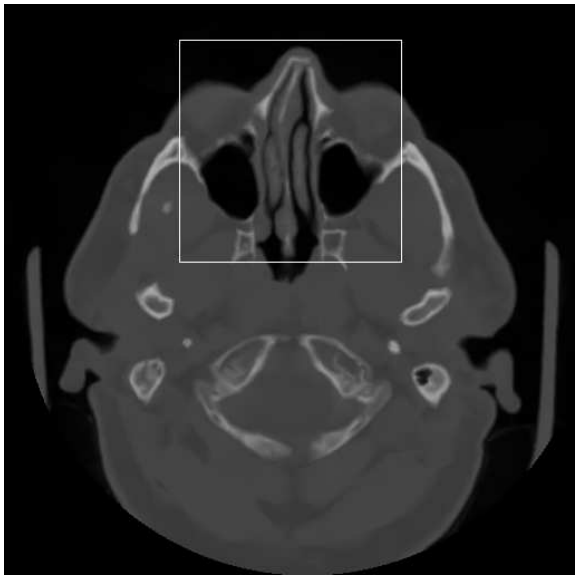
`lena` at sampling density of 2%
(mean and maximum difference 1.50 dB and 4.81 dB)



`peppers` at sampling density of 2%
(mean and maximum difference 1.61 dB and 5.64 dB)

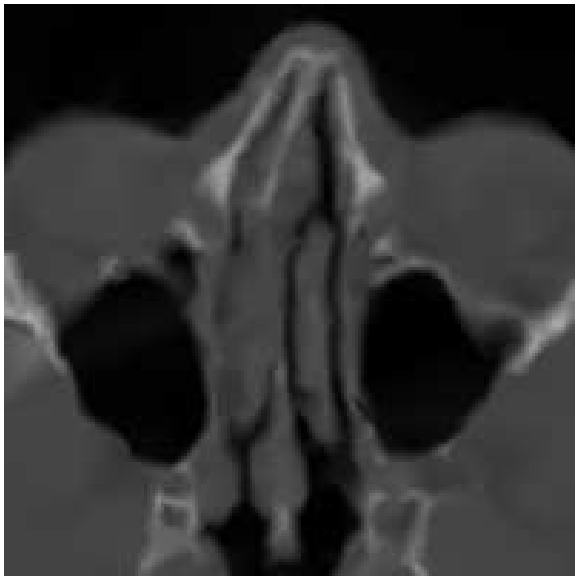
- proposed method consistently yields image reconstructions with higher PSNR (often by several dB) than IT method, except at rates too low to be of practical interest

Lossy Coding Example: `ct` image, 3757 bytes decoded (about 62%)

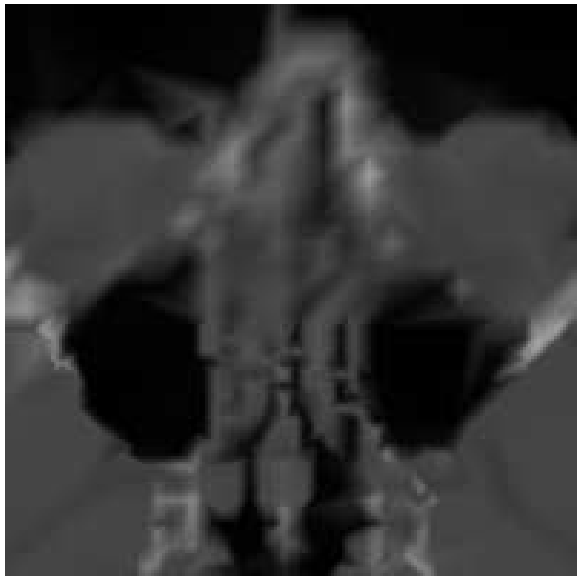


Original Image

Original



Proposed (33.94 dB)



IT (27.59 dB)

Lossy Coding Example: lena image, 6000 bytes decoded (about 64%)

Original Image

Original

Proposed (27.46 dB)

IT (25.07 dB)

Conclusions

- proposed new coding method for arbitrarily-sampled image data
- introduced effective representation for arbitrarily-sampled image data, namely ADIT
- presented means to code this representation
- proposed coding method shown to have vastly superior progressive coding performance at lossy rates relative to state-of-the-art IT method (both in terms of PSNR and subjectively)
- proposed coding method has slightly better coding performance at lossless rates
- of potential benefit to many applications employing arbitrarily-sampled image datasets

Questions?