# 2D-Pattern Matching Image and Video Compression: Theory, Algorithms, and Experiments

Marc Alzina
ENST
46 Rue Barrault
75013 Paris, France.
alzina@email.enst.fr

Wojciech Szpankowski[1] and Ananth Grama[2]
Department of Computer Sciences
Purdue University
W. Lafayette, IN 47907, USA
{spa, ayg}@cs.purdue.edu

[1] This work was supported by NSF Grants NCR-9415491 and C-CR-9804760.
[2] This work was supported by NSF Grants ACI-9872101 and EIA-9806741.

**Acknowledgments**: It is our privilege to acknowledge valuable discussions with Dr. Yuryi Reznik (RealNetwork Inc.) and Prof. I. Konotoyiannis (Purdue University).

# Pattern Matching Framework: Overview

❑ Summary of Results

❑ Theoretical Foundations

- ➔ Fixed Database Model (Video Matching)
- ➔ Growing Database Model (Image Matching)

❑ Algorithmic Aspects and Implementation

- ➔ k-d Trees for Matching
- ➔ Augmented Run-Length Encoding
- ➔ Distortion Metrics

❑ Experimental Studies

## Summary of Contributions

- Develop a lossy data compression framework based on approximate 2-D pattern matching.
- Demonstrate the use of this framework for image and video compression.
  (Higher level schemes based on prediction, frequency domain techniques, and others can be built on top of this framework.)
- Theoretical Contributions:
  - Show that the fixed database model used for video compression leads to suboptimal performance.
  - Provide an approximate analysis for the growing database model used in image compression.
- Implementation of 2D-PMC using a range of innovative algorithmic techniques, data structures, generalized run length coding, adaptive arithmetic coding, and variable and adaptive maximum distortion level.
- Results and Comparisons.

# 2D-PMC: The Basis of Pattern Matching Compression

- The central theme of a lossy extension of Lempel-Ziv algorithm is the notion of approximate repetitiveness.
- If a portion of data recurs in an approximate sense, then subsequent occurrences can be stored as direct or indirect references to the first occurrence.

**Key Challenges:**

- Theoretical Underpinnings
- Algorithms for Approximate Matching
- Appropriate Distortion Measures

# Theoretical Underpinnings:

## Terminology and Assumptions:

- ❑ The source sequence $\{X_k\}_{k=1}^{\infty}$ takes values from a finite alphabet $A$ (e.g. $|A| = 256$). That is, $X_i : S \to A$, where $S$ is a two-dimensional area.

- ❑ Sequence $X_m^n = X_m X_{m+1} \ldots X_n$ and $P(x_1^n)$ is the probability that $X_1^n = x_1^n$.

- ❑ Sequence $X_1^n$ is encoded into a code $c_n$ and the decoder produces an estimate $\hat{X}_1^n$ of $X_1^n$.

- ❑ The code $c_n$ is a function $\Phi : A^n \to \{0, 1\}^*$, thus $c_n = \Phi(x_1^n)$.

- ❑ The decoder applies function $\psi : (0, 1)^* \to A^n$ to find $\hat{x}_1^n = \psi(c_n)$

- ❑ $l(c_n)$ is the length of code representing $c_n$, bit rate $r(x_1^n) = (l(c_n))/n$, average bit rate $E[r(X_1^n)] = (E[l(c_n)])/n$, and compression ratio CR is $(n \log|A|)/(l(c_n))$.

# Theoretical Underpinnings

## Distortion Measures

We consider single-letter fidelity distortion measures $d : A \times A \to \Re_+$ such that

$$d(x_1^n, \hat{x}_1^n) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, \hat{x}_i)$$

Which is assumed to be sub-additive, i.e., for any two integers n, m, and given vectors $x^{n+m}$ and $y^{n+m}$,

$$d(x^{n+m}, y^{n+m}) \le \frac{n}{n+m} d(x_1^n, y_1^n) + \frac{m}{n+m} d(x_{n+1}^{m+n}, y_{n+1}^{m+n})$$

Examples of such distortion measures include the Hamming distance and Squared-error distortion.

# Theoretical Underpinnings

**Fixed Database Model (basis for video matching framework)**

- Both decoder and encoder have access to the common database sequence $\hat{X}_1^n$ generated according to distribution $\hat{P}$.

- The source sequence $X_1^M$ (where $M = N^2$ for an $N \times N$ image) is partitioned according to $\Pi_n$ into variable length phrases (rectangles in 2D-PMC) $Z^1, Z^2, \ldots, Z^{|\Pi_n|}$ of lengths $L_n^1, L_n^2, \ldots, L_n^{|\Pi_n|}$ respectively. Here, for given distortion bound $D > 0$,
$L_n^1 = max\left\{k;(d(\hat{X}_i^{i+k-1}, X_1^k) \le D)\right\}$ and $Z^1 = X_1^{L_n^1}$.

- This implies that the first partition comprises of the longest prefix of input $X_1^M$ that matches a substring in database $\hat{X}_1^n$ to specified tolerance.

- The corresponding string recovered by the decoder is $\hat{Z}^1 = \hat{X}_i^{i + L_n^1 - 1}$

# Theoretical Underpinnings

## Fixed Database Model (basis for video matching framework)

Subsequent partitions are computed in a similar manner:

$$L_n^{m+1} = max\left\{k;\left(d\left(\hat{X}_i^{i+k-1}, X_{K_n(m)+1}^{K_n(m)+k}\right) \le D\right)\right\}$$

$$Z^{m+1} = X_{K_n(m)+1}^{K_n(m)+k}$$

$$\hat{Z}^{m+1} = \hat{X}_i^{i+L_n^m-1}$$

where

$$K_n(m) = \sum_{i=1}^{m} L_n^i$$

is the accrued length of coded segments.

# Theoretical Underpinnings

**Fixed Database Model (basis for video matching framework)**

❑ The source sequence is partitioned into $Z^1, Z^2, \ldots, Z^{|\Pi_n|}$ and the decoder recovers $\hat{Z}^1, \hat{Z}^2, \ldots, \hat{Z}^{|\Pi_n|}$. This is within distortion $D$ from $X_1^M$ because of the subadditive property of the distortion measure.

❑ Each $\hat{Z}^i$ is represented by a pointer to the database and its length, hence its description costs $O(\log(n) + \log(L_n^i))$ bits. The total compressed code length is

$$l_n(X_1^M) = \sum_{i=1}^{|\Pi_n|} \log(n) + \Theta(\log(L_n^i))$$

and the bit-rate is given by

$$r_n(X_1^M) = \frac{1}{M} \sum_{i=1}^{|\Pi_n|} \log(n) + \Theta(\log(L_n^i))$$

# Theoretical Underpinnings

**Fixed Database Model (basis for video matching framework)**

To formulate our main result, we introduce the generalized Shannon entropy

$$\hat{r}_0(D) = \lim_{n \to \infty} \frac{E_P[-\log \hat{P}(B_D(X_1^n))]}{n}$$

where $B_D(\hat{x}_1^n) = \left\{ y_1^n ; (d(\hat{x}_1^n, y_1^n) \le D) \right\}$ is the ball of radius $D$ and center $\hat{x}_1^n$ and $E_P$ represents expectation with respect to $P$.

# Theoretical Underpinnings

**Fixed Database Model (basis for video matching framework)**

**Theorem 1**: Consider the fixed database $\hat{X}_1^n$ generated by a Markovian source $\hat{P}$ and source sequence $X_1^M$ generated by Markovian source $P$.

➔ The length $L_n^1$ of the first matched phrase satisfies

$$\lim_{n \to \infty} \frac{L_n^1}{\log n} = \frac{1}{\hat{r}_0(D)}$$

More precisely, for any $\varepsilon > 0$,

$$Pr\left\{ (1-\varepsilon)\frac{1}{\hat{r}_0(D)}\log n \le L_n^1 \le (1+\varepsilon)\frac{1}{\hat{r}_0(D)} \right\} = 1 - O\left(\frac{\log n}{n^{\varepsilon}}\right)$$

➔ The average bit rate asymptotically attains

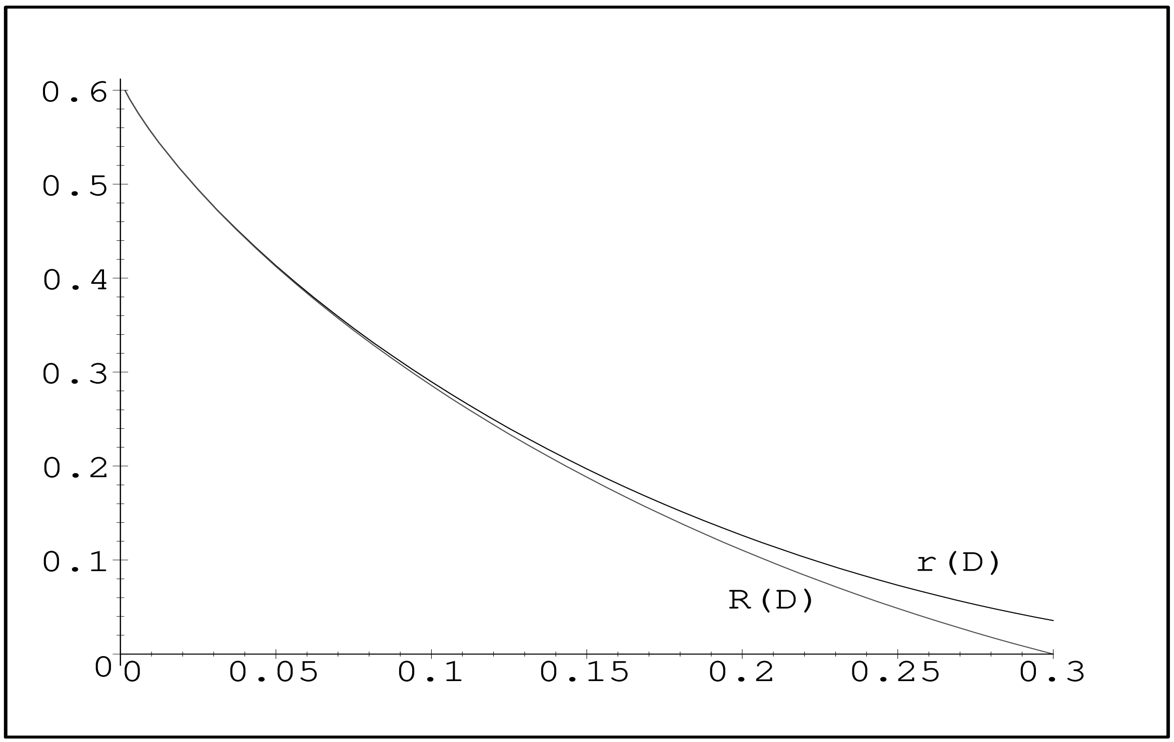
$$\lim_{n \to \infty} \lim_{M \to \infty} E_P[r_n(X_1^M)] = \hat{r}_0(D)$$

(see paper for proof).

# Theoretical Underpinnings

**Fixed Database Model (basis for video matching framework)**

Comments: Comparison optimal rate distortion R(D) and the generalized Shannon entropy r(D) for memoryless binary source.



# Theoretical Underpinnings

## Growing Database Model (basis for image matching framework)

❏ In contrast to video matching, image matching uses the most recent partially compressed subimage as the database.

❏ We use a Growing Database Model for image matching.

❏ The source sequence $\{X_k\}_{k=1}^{\infty}$ is generated according to the distribution $P$.

❏ The initial reproduction sequence of length $m$ is set to be the same as the first $m$ symbols in the database, i.e., $X_1^m = \hat{X}_1^m$ and reveal $X_1^m$ to the decoder losslessly.

❏ This is followed by the matching and the database growing process.

# Theoretical Underpinnings

## Growing Database Model (basis for image matching framework)

- Set $\hat{X}^0 = X_1^m$ and let $n_0 = m$.
- The first match of length $L_{n_0}^1$ in the database is determined as follows:

$$L_{n_0}^1 = max\left\{t;\left(\left(d\left(\hat{X}_i^{i+t-1}, X_{n_0}^{n_0+t-1}\right) \le D\right), 1 \le i \le n_0 - t\right)\right\}$$

- The database is enlarged as follows:

$$\hat{X}^1 = \hat{X}^0 \oplus \hat{X}_i^{i+t-1}$$

- Here, $a \oplus b$ represents the concatenation of $a$ and $b$. Note that we do not concatenate $X_{n_0}^{n_0+t-1}$, rather the distorted database $\hat{X}_i^{i+t-1}$. This is so that errors do not propagate.

# Theoretical Underpinnings

## Growing Database Model (basis for image matching framework)

- The matching process occurs as follows: provided the $k^{\text{th}}$ reproduction sequence $\hat{X}^k$ of length $n_k = |\hat{X}^k|$ was constructed, define the $(k+1)$st match length as:

$$L_{n_k}^{k+1} = max\left\{t; \left(\left(d\left(\hat{X}_i^{i+t-1}, X_{n_k}^{n_k+t-1}\right) \le D\right), 1 \le i \le n_k - t\right)\right\}$$

- The database is enlarged to:

$$\hat{X}^{k+1} = \hat{X}^k \oplus \hat{X}_i^{i+t-1}$$

- As before, we update the database with the distorted segment, thus ensuring that the errors dont propagate.

- The operational bit-rate at step k of the algorithm is given by:

$$r_m\left(X_m^{n_k}\right) = \frac{\log n_o + \log n_1 + \ldots + \log n_{k-1} + \Theta\left(\log L_{n_0}^1\right) + \ldots + \Theta\left(\log L_{n_{k-1}}^k\right)}{L_{n_0}^1 + L_{n_1}^2 + \ldots + L_{n_{k-1}}^k}$$

# Theoretical Underpinnings

## Growing Database Model (basis for image matching framework)

❏ The analysis of $r_m\left(X_m^{n_k}\right)$ is difficult because the distorted database $\hat{X}^k$ is non-stationary of unknown distribution. We propose a heuristic and approximate analysis for large $m$ and

$$k = o((\log m)/(\log\log m))$$

❏ We start with the observation that the growing databases $\hat{X}^0, \hat{X}^1, \ldots, \hat{X}^k$ are non-stationary with probability measures $\hat{P}^0, \hat{P}^1, \ldots, \hat{P}^k$, provided they exist.

❏ We postulate (based on Theorem 1) that $L_{n_k}^{k+1}$ is of the order of $\frac{1}{\hat{r}_0^k(D)}\log n_k$, where the coefficient $\frac{1}{\hat{r}_0^k(D)}$ depends on probability measure $\hat{P}^k$.

# Theoretical Underpinnings

## Growing Database Model (basis for image matching framework)

**Hypothesis**: At step $k$ of the algorithm, with database size $n_k$ and underlying probability $\hat{P}^k$, there exists a constant $\hat{r}_0^k(D)$ which depends on $\hat{P}^k$ such that in probability and on average

$$L_{n_k}^{k+1}(\hat{P}^k) \sim \frac{1}{\hat{r}_0^k(D)} \log n_k$$

for $n_k \to \infty$.

It is important to note that this hypothesis must be rigorously proven if one hopes for a precise analysis of the growing database model.

Under this hypothesis, we can show that:

$$E\left[r_m\left(X_m^{n_k}\right)\right] \approx \frac{k}{\frac{1}{\hat{r}_0^0(D)} + \frac{1}{\hat{r}_0^1(D)} + \dots + \frac{1}{\hat{r}_0^{k-1}(D)}}$$

provided $k = o((\log m)/(\log\log m))$ and $m \to \infty$.

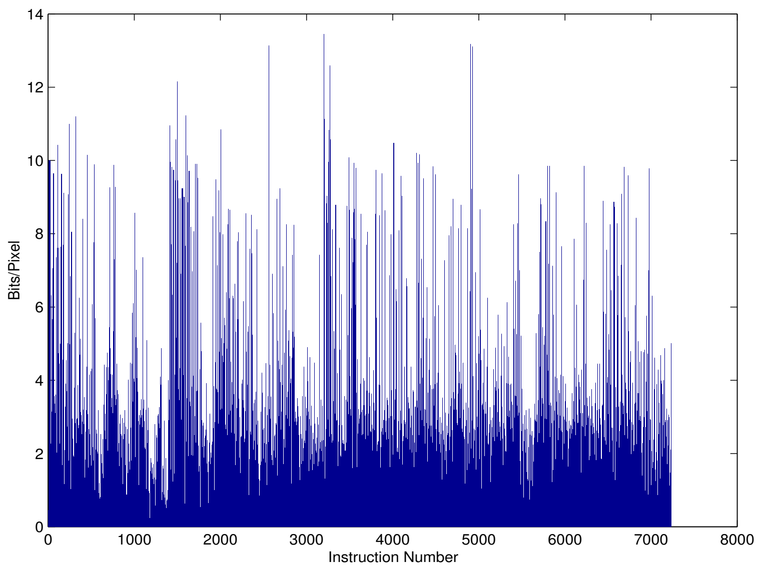
# Theoretical Underpinnings

## Growing Database Model (basis for image matching framework)

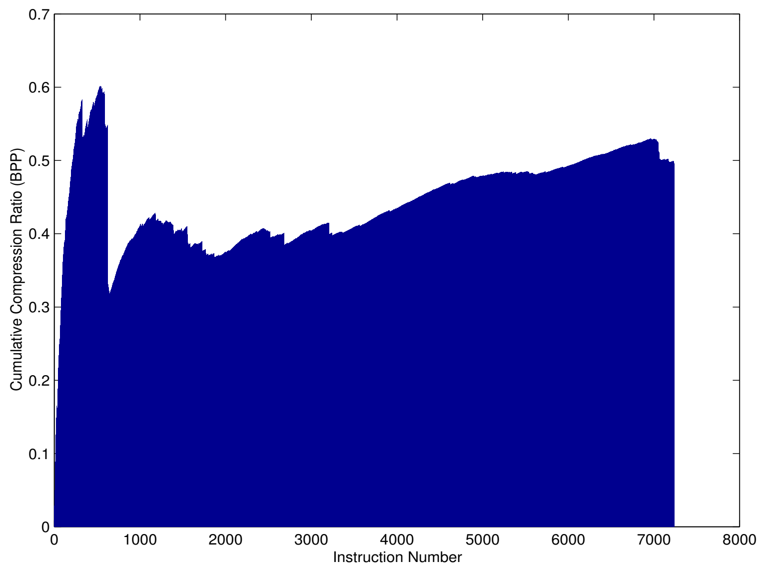
❑ The analysis rests on our hypothesis; whose validity depends on how non-stationary the database is.

❑ Our preliminary theoretical investigation suggests that it is possible to construct a non-stationary database such that the hypothesis does not hold.

❑ Even if the hypothesis holds, the compression ratio may not converge (as is illustrated below for sample image Basselope at 0.5 BPP.




Compression Ratio as database grows




BPP as database grows.

# Algorithms, Data Structures, and Implementation

2D-PMC has three encoding mechanisms:

- ❏ 2D Pattern Matching
- ❏ Enhanced Run-Length Encoding
- ❏ Lossless Coding.

# Algorithms, Data Structures, and Implementation

## 2D Pattern Matching

- The basic idea is to find a 2D region (rectangle) in the uncompressed part that occurs approximately in the compressed part (the database). The notion of approximate occurrence is formalized when discussing distortion measures.

- Since the objective is to find the largest such area, brute-force methods are prohibitively expensive.

- We need a range search to detect approximate occurrences as opposed to an exact search.

- This search is performed in two steps:

  - Identify candidate seed points for matches

  - Grow regions around selected points and select largest such region.

# Algorithms, Data Structures, and Implementation

## Identifying seed points for matches

- Define an anchor point as the top left corner of the segment for which a match is being sought.

- We consider a template of $k$ pixels around the anchor point (for example this template could be a 2 x 3 region).

- The problem reduces to searching a 6-dimensional space (2x3=6) in an approximate sense for all vectors that are in the neighborhood of this template.

- We use a k-d tree for performing this search.

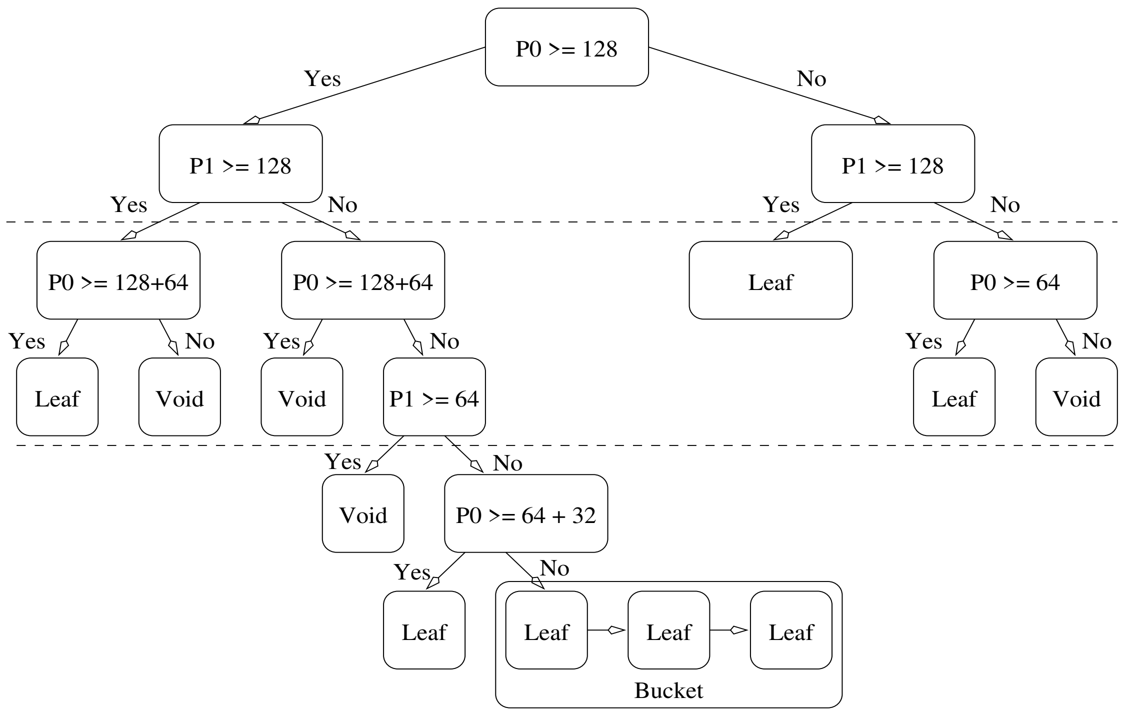
**Note:** A 1-D tree is an ordinary binary search tree; a 2-d tree is similar, but the nodes at even levels compare x-coordinates and the nodes on odd-levels compare y-coordinates when branching. In general, a k-d tree has nodes with k coordinates, and branching at each level is based on only one of the coordinates; for example, we might branch on coordinate number (k mod l)+1 on level l.

# Algorithms, Data Structures, and Implementation

**Identifying seed points using k-d trees.**

❑ To search a k-dimensional vector using k-d trees, we trace the tree from the root down to one of the terminal nodes. The decision whether to go left or right at a node is performed on the (k mod l)-th coordinate, where l is the depth of the node.

❑ The terminal nodes are called buckets since they contain one or more leaves. Leaf nodes store data corresponding to the vector and coordinates of the associated pixels.



# Algorithms, Data Structures, and Implementation

**Building k-d trees.**

To insert a leaf L, we start from the root and follow the path until we find a bucket B. There are three possibilities:

- ❑ Bucket B is empty; if so, we put L into it.
- ❑ The depth of this branch is already equal to the maximum depth MaxDepth. In this case, we simply add L to the bucket which is maintained as a chain of leaves.
- ❑ The depth of this branch is less than the limit. In this case, we replace B by a node whose children are B and whose bucket is empty. We then continue the insertion process.

# Algorithms, Data Structures, and Implementation

## Approximate Search in k-d Trees.

- We do a range search on a k-d tree with two arguments: a minimum vector $(m_1, m_2, ..., m_k)$ and a maximum vector $(M_1, M_2, ..., M_k)$.

- These vectors are determined by the distortion measures.

- We search for k-dimensional vectors $(v_1, v_2, ..., v_k)$ such that $m_1 <= v_1 <= M_1, ...,$ $m_k <= v_k <= M_k$. This is done recursively by keeping every subtree that may contain a solution and discarding the others.

# Algorithms, Data Structures, and Implementation

**Complexity of k-d tree search.**

- The k-d tree search used here falls under the model of Friedman et al.

- Let us bound the number of paths explored per coordinate assuming the maximum depth is $d \sim \log N$, the search range is $[\alpha, \beta]$ for every coordinate, and entries take values between $a$ and $b$.

- The maximum number of subranges that overlap with the search range is: $s = \left\lceil \frac{2^d(\beta - \alpha)}{b - a} \right\rceil + 1$. Thus, for a problem of dimension $k$, we must search at most $s^k$ buckets.

- A bound for the maximum number of comparisons is $q = O(ks^k d)$.

- For example, for $b - a = 255$, $\beta - \alpha = 8$, $d = 5$, and $k = 6$ we have $s = 2$ and $q = 1953$.

- Actual search performance of a k-d tree is illustrated by the example: $k = 6$, database size: 262144 = 512 x 512 random vectors (8 bits per coordinate), maximum tree depth: 30 (5 levels per coordinate), solutions found: 72, search time: 2.7ms (8.11s for 3000 searches) on a PII 300MHz.

# Algorithms, Data Structures, and Implementation

**Optimizations for k-d search.**

- In order to find all interesting solutions, we actually use several k-d trees, each one associated with a different template.

- To decrease the number of potentially non-interesting solutions, pixels and their associated vectors are grouped by locations (by rectangles whose typical size is 30 x 30) and one k-d tree is used per location, so that we can perform a local search.

Our implementation of k-d tree searches follows the work of Constantinescu and Storer [1994].

# Algorithms, Data Structures, and Implementation

**Enhanced Run-Length Encoding (ERLE).**

- We extend RLE using a simple variant called enhanced run-length encoring.
- ERLE fits a planar surface on to the given m x n grid of pixel values.
- The coefficients of the planar surface are determined simply by solving a system of normal equations to minimize least-squared error.
- Once the planar surface has been determined, the sub-segment of the m x n grid that is within the distortion envelope can be identified and coded using ERLE.
- The maximum such region positioned at the anchor point is computed by recursively doubling the size until we exceed the distortion envelope.

The technique (2D-PMC or ERLE) yielding lower cost is used to code the segment.

# Algorithms, Data Structures, and Implementation

**Lossless coding.**

- ❑ Once the lossy image compression is completed, the image description is a script in the following form: `instruction1 parameters1 instruction2 parameters2` and so on.
- ❑ Instructions can be one of {`pixel_color, ERLE, match_pointer`}.
- ❑ This stream is sent to the arithmetic coder. Since the probabilities of instructions and parameters are not identical, we can get additional savings by using an arithmetic coder.
- ❑ This coder is built around the ideas of Howard and Vitter [DCC91].

# Distortion Measures

In 2D-PMIC we assign to each individual pixel a range (i.e., lower and upper bounds on the allowable distortion value per pixel) that depends on the values of surrounding pixels. The basic idea is to be less tolerant (i.e., have a small range) in slowly varying regions and more tolerant in quickly varying regions.

## Local Distortion Range

```
Range=RandomNoise + LocalNoise x LocalVariation
```

Here, `RandomNoise` corresponds to uniform distortion, `LocalNoise` is additional distortion, and `LocalVariation` is the maximum of the absolute value of the differences between the pixel values at `(x,y)` and its 8 neighbors.

The local nature of this scheme makes it sensitive to noise.

## Global Distortion Range

To reduce the sensitivity of local distortion range, we increase the scope beyond the neighbors to a circular region and weight differences with distance from source point.

This yields better results visually (less specky images) but introduces additional tuning parameters.

# Overall Space and Time Complexity

❏ The compression complexity is optimal and equal to $O(N^2)$, the decompression complexity is $O(N^2/\log N)$, and space complexity is $O(N^2)$.

❏ The constant associated with compression is much higher than that for decompression (as we shall see).

❏ To get an idea of actual memory requirements, the k-d trees for Lena, Basselope, and Banner require 31.6 MB, 21MB, and 2MB respectively.

| Image | BPP | Compr. Time | Decompr. Time |
|---|---|---|---|
| Lena | 0.25 | 159 | 0.075 |
| | 0.5 | 187 | 0.153 |
| Basselope | 0.27 | 181 | 0.106 |
| | 0.5 | 172 | 0.198 |
| San Francisco | 0.25 | 149 | 0.061 |
| | 0.5 | 165 | 0.125 |
| Banner | 0.25 | 11 | 0.007 |
| | 0.5 | 10 | 0.010 |

Compression and decompression times (in seconds) for images.

## Implementation Notes for Video Compression

❑ Pattern matching video compression uses the previous compressed frame in its uncompressed form as the database to compress current frame.

❑ This is make sure errors do not propagate during compression.

❑ Applying this framework directly to video compression, it was noticed that often the matched patterns were often quite small leading to low compression ratios. This was alleviated by matching a low-pass filtered version of the image. References are still made to the unfiltered image. This improved compression ratios significantly.

| Image# | BPP | CR | RMSE | PSNR | BPP | CR | RMSE | PSNR |
|---|---|---|---|---|---|---|---|---|
| | Unfiltered Matching | | | | Filtered Matching | | | |
| 2 | 0.57 | 14.0 | 16.8 | 23.64 | 0.09 | 84.9 | 20.6 | 21.86 |
| 3 | 0.36 | 22.3 | 19.3 | 22.43 | 0.11 | 73.9 | 23.6 | 20.68 |

# Experimental Results

We demonstrate from our experiments that:

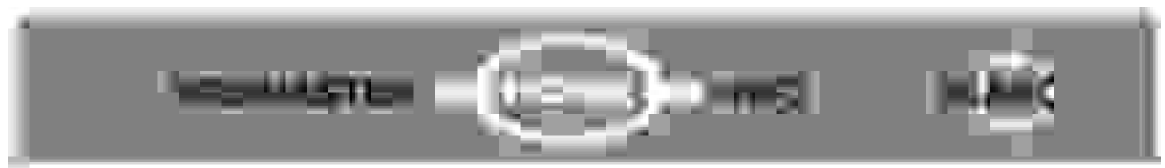
- ❏ For synthetic images (Banner and Basselope), 2D-PMIC outperforms JPEG both in terms of compression ratios as well as quality.
- ❏ For compression of general images, 2D-PMIC is competitive with JPEG and wavelet compression up to 0.25 bpp.
- ❏ At low bit rates, frequency domain methods are still better than 2D-PMIC for natural images.

Our results indicate that for synthetic images and higher bit rates, 2D-PMIC is the method of choice, particularly because of the simplicity of the decompression procedure.

# Experimental Results

2D-PMIC
Banner 0.286 bpp
PSNR  28.59 dB


JPEG
Banner 0.288 bpp
PSNR 19.39

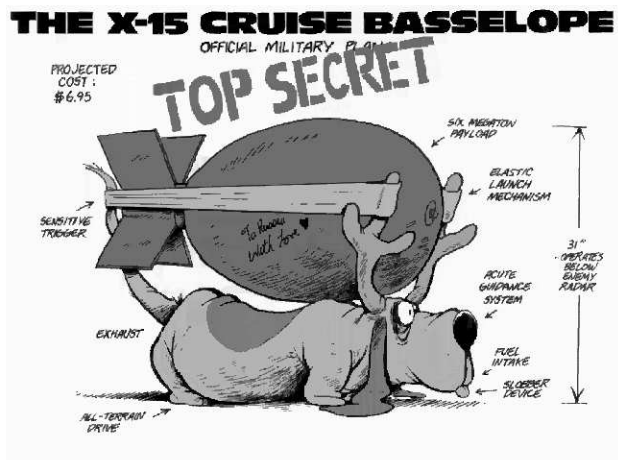


2D-PMIC
Banner 0.500 bpp
PSNR 45.77

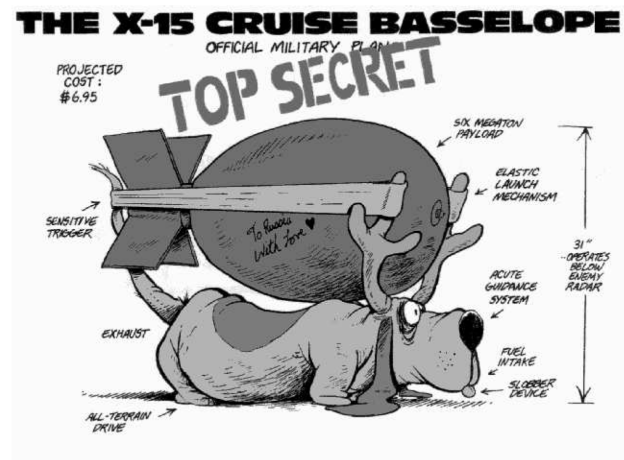

JPEG
Banner 0.508 bpp
PSNR 24.46

# Experimental Results

2D-PMIC

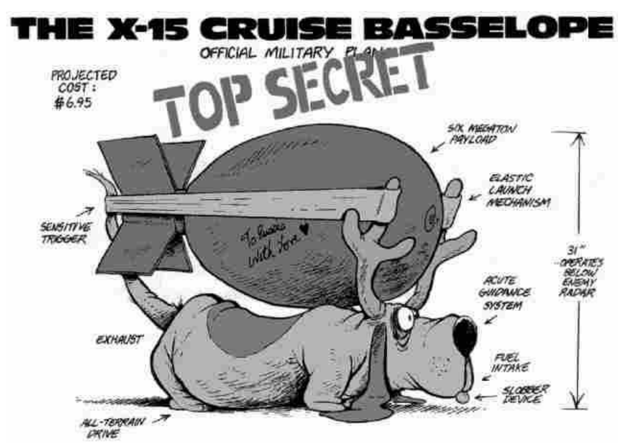



0.513bpp, PSNR 26.15




0.95bpp (Lossless)

JPEG




0.495bpp, PSNR 26.17

1.007bpp, PSNR 31.35

# Experimental Results

2D-PMIC

0.499bpp, PSNR 25.77

1.05bpp, PSNR 31.61

JPEG

0.503bpp, PSNR 27.61

1.00bpp, PSNR 31.53

# Experimental Results

2D-PMIC
0.489bpp
29.30 PSNR

1.051bpp
33.09 PSNR

JPEG
0.499bpp
32.87 PSNR

1.000bpp
35.70 PSNR

# Experimental Results: Comparison with JPEG

| 2D-PMIC | | | | JPEG | | | |
|---|---|---|---|---|---|---|---|
| BPP | CR | RMSE | PSNR | BPP | CR | RMSE | PSNR |
| Image: Banner | | | | | | | |
| 0.29 | 27.98 | 9.5 | 28.6 | 0.29 | 27.80 | 27.4 | 19.4 |
| 0.50 | 15.99 | 1.3 | 45.8 | 0.50 | 15.87 | 15.3 | 24.5 |
| 0.54 | 14.89 | 0.0 | Inf | 1.01 | 7.94 | 15.1 | 24.6 |
| | | | | 2.00 | 4.00 | 15.1 | 24.6 |
| Image: Basselope | | | | | | | |
| 0.27 | 29.56 | 21.0 | 21.7 | 0.25 | 32.31 | 19.3 | 22.4 |
| 0.51 | 15.58 | 12.6 | 26.2 | 0.50 | 16.17 | 12.5 | 26.2 |
| 0.96 | 8.33 | 0.0 | Inf | 1.00 | 7.95 | 6.9 | 31.4 |
| | | | | 2.01 | 4.08 | 2.6 | 39.7 |
| Image: Lena | | | | | | | |
| 0.25 | 32.01 | 10.8 | 27.5 | 0.25 | 32.30 | 8.9 | 29.1 |
| 0.49 | 29.30 | 8.7 | 29.3 | 0.50 | 16.03 | 5.8 | 32.9 |
| 1.05 | 7.61 | 5.6 | 33.1 | 1.00 | 8.04 | 4.2 | 35.7 |
| 1.94 | 4.13 | 3.6 | 37.1 | 2.01 | 3.81 | 2.7 | 39.4 |
| Image: San Francisco | | | | | | | |
| 0.25 | 32.58 | 17.0 | 23.5 | 0.25 | 31.90 | 15.5 | 24.3 |
| 0.50 | 16.09 | 13.1 | 25.8 | 0.50 | 15.91 | 10.6 | 27.6 |
| 1.05 | 7.59 | 6.7 | 31.6 | 1.00 | 8.02 | 6.8 | 31.5 |
| 2.03 | 3.95 | 2.9 | 38.8 | 2.01 | 3.98 | 3.5 | 37.2 |

# Experimental Results: Video Compression

| Video | Mbps | CR | PSNR |
|---|---|---|---|
| Clairehq | 0.14 | 169 | 33.7 |
| Clairelq | 0.07 | 347 | 31.0 |
| Ping-Ponghq | 0.40 | 58 | 24.0 |
| Ping-Ponglq | 0.26 | 89 | 22.6 |

Video compression results (HQ=high quality and LQ=low Quality).

# Concluding Remarks

- 2D-PMC yields excellent compression ratios and is competitive with JPEG up to 0.25 bpp. 2D-PMIC works better on synthetic images while transform-based methods work better on natural images.

- 2D-PMC is slower in terms of compression time but extremely fast and simple in terms of decompression. Decompression can be done on-fly with minimal software support at the client.

- The pattern matching framework of 2D-PMIC can be used in conjunction with many of the current techniques to leverage the benefits these schemes.

**Work in Progress**

- Improving the matching framework both in terms of speed and quality.

- Accurate analysis of the Growing Database Model.