Fast Planar-Oriented Ripple Search Algorithm for Hyperspace VQ Codebook

Chin-Chen Chang, Fellow, IEEE, and Wen-Chuan Wu
IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 16, NO. 6, JUNE 2007

Presenter: Tzu-Meng Huang
Adviser: Dr. Yeou-Jiunn Chen
Date: 2011/3/30
Outline

• Introduction
• Previous Works
• Proposed Planar Voronoi Diagram Search Method (PVDS)
• Experimental Results
• Conclusion
• Future works
• References
INTRODUCTION
Vector Quantization

• Simply Definition
  • A mapping function which maps a $k$-dimensional vector space $S_k$ into a finite subset $\Psi = \{C_1, C_2, \ldots, C_N\}$

• Codebook
  • Set $\Psi$ holding vectors $N$

• Codeword
  • Vector $C_i = \{c_{i1}, c_{i2}, \ldots, c_{ik}\}$ in $\Psi$
  • $k$-dimensional
Full-search VQ (FSVQ)

- The best matching codeword
- The squared Euclidean distance

\[ d(X, C_{bm}) = \min_{1 \leq i \leq N} \{ d(X, C_i) \} \text{, where} \]

\[ d(X, C_i) = \| X - C_i \|^2 = \sum_{j=1}^{k} (x_j - c_{ij})^2 \]

\[ O(k \times N) \]
VQ-based techniques

- A high-cost quantizer design restricts the practicability of VQ
- VQ-based techniques
  - Full-search equivalents
    - PDS, DTA, EVM, LUT, and others
  - Partial-search methods
    - Tree-based
    - Projection-based structures
Purpose

- A new partial-search method to speed up the complicated quantization process of the traditional VQ.
- Building on the basis of planar Voronoi diagram projection.
PREVIOUS WORKS
DTPC

• Double Test of Principal Components Encoding Method (DTPC)
  • principal component analysis (PCA)
  • To reduce the hyperspace vector space to a low-dimensional space.
• The lower the number of vector dimensions is, the less effort is required to do distance evaluation.
Principal Component Analysis (PCA)

- Finding a linear transformation that can project each \( k \)-dimensional vector into an \( s \)-dimensional space (\( s \leq k \)).

- **Step 1**
  - Calculate the co-variance matrix of these \( N \) vectors in the codebook.

- **Step 2**
  - Discover all the eigen-values \( \lambda_1, \lambda_2, \ldots, \lambda_k \) of this matrix and their corresponding eigen-vectors \( D_1, D_2, \ldots, D_k \).
  - \( D_1 \) is capable of preserving the most variation among the original vectors in the projected values.
DTPC

- Adopting axes $D_1$ and $D_2$ to project all the codewords onto a plane before the quantization phase.

- **Preprocessing Procedure**

  - **Step 1** Find the projection axes $D_1$ and $D_2$ of codebook $\Psi$.
  - **Step 2** Project each codeword $C_i$ to a planar point $pC_i(\tilde{c}_{i1}, \tilde{c}_{i2})$ by $\tilde{c}_{ij} = \sum_{l=1}^{k} c_{il} \times d_{jl}$, where $d_{jl}$ is the $l$th element of axis $D_j$, $i = 1, 2, \ldots, N$, and $j = 1, 2$.
  - **Step 3** Sort these projected values $\tilde{c}_{i1}$ of the codewords in ascendant order and record them in an array $A$. 

2011/3/24
DTPC

• Vector Searching Procedure

Step 1) Project input vector $X$ to a point $p_X(\tilde{x}_1, \tilde{x}_2)$ by $\tilde{x}_j = \sum_{l=1}^{k} x_l \times d_{jl}$, where $j = 1, 2$.

Step 2) Seek the closest projected value with respect to $\tilde{x}_1$ by using binary search from the array $A$ and take it as the initial codeword.

Step 3) Label a search domain including $TH$ codewords from the neighbors of the initial codeword, where $TH \ll N$.

Step 4) Compute the squared Euclidean distances between vector $X$ and the codewords inside the search domain in order by using EVM, DT, and LUT techniques. Finally, derive the appropriate one to encode.
DTPC

Fig. 1. Example of the search domain in the DTPC method.
To save the cost of time, binary search has been tested in place of linear search.

TSVQ
- Tree structure
- The reconstructed content tends to be severely distorted.
- The nonleaf nodes (virtual codewords) of the tree structure cannot sufficiently stand for their children.

Hierarchy-Oriented Searching Method (HOSM)
- Unbalanced-binary-tree LBG (UBT-LBG)
UBT-LBG Procedure

Step 1) Take the input codewords as a training set \( T \).
Step 2) Derive a centroid \( G \) of the set \( T \). According to the centroid \( G \), calculate two virtual codewords (VCs) by using \( g_1 = G \times (1 - 0.01) \) and \( g_2 = G \times (1 + 0.01) \).
Step 3) Partition the set \( T \) into two subsets based on VCs \( g_1 \) and \( g_2 \) by using the LBG algorithm.
Step 4) Choose a subset of the set \( T \) that has the most codewords and then process it again through steps from 2 to 4 until the number of VCs reaches what we desire, where each codeword of the input is dominated by one VC.
Step 5) Apply the LBG algorithm again to improve these VCs by using the input.
HOSM

- Convert the codebook into the hierarchical structure.
- A top-down path
- Compares the codewords in a dominated set at each level and selects the closest.
- A duplication mechanism at the bottom level
  - reduce the probability of distortion resulting from misplacement
HOSM

Step 1) Take codebook $\Psi$ with $N$ codewords as a set $\Psi_t$ to be processed.

Step 2) Do UBT-LBG to $\Psi_t$ to generate $|\Psi_t|/4$ VCs. Then, these VCs are taken as $\Psi_t$. Construct a hierarchy according to the dominative relations.

Step 3) Go back to Step 2) until the size of $\Psi_t$ is smaller than four. Then, the hierarchical tree is constructed.

Step 4) Divide the training samples into $k$-dimensional vectors and put them in a pool.
Step 5) Get a block from the training pool. Apply the FSVQ algorithm to find the best matching codeword \( C_{bm} \) with respect to the block.

Step 6) Take the top-down dominative-area path and search for the appropriate codeword \( C_{fd} \). If \( C_{bm} \) is not in the dominative area where \( C_{fd} \) is situated, make a copy of \( C_{bm} \) in this dominative area.

Step 7) Repeat Steps 5) and 6) until all the blocks in the pool are through.
Fig. 2. Example of the hierarchical tree structure in HOSM.
HOSM

Vector Searching Procedure

Step 1) Put all the VCs at the top level in a set $\Psi_e$.

Step 2) Find the closest VC in set $\Psi_e$ with respect to the input vector $X$ by computing and comparing the squared Euclidean distance. Then, take the area dominated by the found VC as set $\Psi_e$.

Step 3) Repeat Step 2) until set $\Psi_e$ goes to the bottom of the tree.

Step 4) Search set $\Psi_e$ again to locate the appropriate codeword to quantize vector $X$. 
PROPOSED PLANAR VORONOI DIAGRAM SEARCH METHOD (PVDS)
Voronoi-Diagram Construction

• Voronoi diagram
  • An implicit geometric interpretation of the nearest neighbors of objects in the space.
  • Can be constructed in any $k$–dimensional space
  • Voronoi cell

Voronoi-Diagram Construction

Take \( P_{c_1}, P_{c_2}, \ldots, P_{c_N} \) as the generators of a Voronoi diagram.

For each point \( P_{c_i} \):

\[
VNC_{\psi}(p_{c_i}) = \bigcup_{i \neq j} \{ p_X \in \mathbb{R}^2 \mid d(p_X, p_{c_i}) < d(p_X, p_{c_j}) \}
\]

- A 2-D space by using the PCA technique in order to reduce the search complexity.
Fig. 3. Construction process of the planar Voronoi diagram.
Preprocessing and Training Procedure

Step 1) Project codebook $\Psi$ on a 2-D space by using the PCA technique.

Step 2) Build a VD out of the points on the plane and construct its adjacency-list structure by creating a one-round neighbor-ripple.

Step 3) Create an LUT lookup table.

Step 4) Select some training samples and divide them into $k$-dimensional vectors.

Step 5) For each training vector, use the FSVQ encoder to derive the best matching codeword $C'_{bm}$.

Step 6) Project each training vector onto the plane and find its initial codeword from VD. If codeword $C'_{bm}$ does not fall in the two-ripple domain of the initial codeword, duplicate codeword $C'_{bm}$ and put it in the adjacency list of the initial codeword.

Step 7) Repeat Steps 5) and 6) until all the vectors are through and get a new VD structure.
Adjacency list of the Voronoi diagram

Fig. 5. Adjacency list of the Voronoi diagram from Fig. 4.
Vector Searching Procedure

Step 1) Project the input vector $X$ onto the same plane by using the PCA technique.
Step 2) Determine the cell that vector $X$ is located in and identify the initial codeword.
Step 3) According to the initial codeword, set a search domain from the VD structure by performing the ripple-like extension operation.
Step 4) Search the domain for the appropriate codeword by computing the squared Euclidean distance through looking up the LUT table. Finally, the appropriate codeword with the shortest distance can be found and used to quantize vector $X$. 
Fig. 4. Example of planar Voronoi diagram with 13 points.
EXPERIMENTAL RESULTS
Experiments

- Six 512 x 512 standard test images
- VQ codebooks
- LBG clustering algorithm
- Codeword
  - A vector with 16 dimensions
- Environment
  - A PC with an AMD 1.31-GHz processor, 256-MB main memory
  - A Borland C++ Builder compiler.
How many ripples is the best for an effective and efficient quantizer?

<table>
<thead>
<tr>
<th>Images</th>
<th>PSNR (dB)</th>
<th>Time (sec.)</th>
<th>Count</th>
<th>0 ripple</th>
<th>1 ripple</th>
<th>2 ripples</th>
<th>3 ripples</th>
<th>4 ripples</th>
<th>5 ripples</th>
<th>6 ripples</th>
<th>FSVQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lena</td>
<td>26.910</td>
<td>0.111</td>
<td>1</td>
<td>30.920</td>
<td>6.9</td>
<td>31.254</td>
<td>19.2</td>
<td>31.313</td>
<td>37.5</td>
<td>62.4</td>
<td>95.2</td>
</tr>
<tr>
<td>Boat</td>
<td>24.990</td>
<td>0.142</td>
<td>1</td>
<td>28.853</td>
<td>6.8</td>
<td>29.227</td>
<td>18.8</td>
<td>29.295</td>
<td>37.3</td>
<td>62.9</td>
<td>95.1</td>
</tr>
<tr>
<td>Jet</td>
<td>25.797</td>
<td>0.141</td>
<td>1</td>
<td>30.085</td>
<td>6.3</td>
<td>30.438</td>
<td>17.9</td>
<td>30.504</td>
<td>34.7</td>
<td>56.3</td>
<td>86.4</td>
</tr>
<tr>
<td>Pepper</td>
<td>26.931</td>
<td>0.109</td>
<td>1</td>
<td>30.371</td>
<td>6.7</td>
<td>30.643</td>
<td>18.5</td>
<td>30.690</td>
<td>36.4</td>
<td>60.4</td>
<td>91.7</td>
</tr>
<tr>
<td>Baboon</td>
<td>21.441</td>
<td>0.156</td>
<td>1</td>
<td>23.626</td>
<td>7.2</td>
<td>23.943</td>
<td>20.6</td>
<td>24.043</td>
<td>41.6</td>
<td>69.8</td>
<td>105</td>
</tr>
<tr>
<td>GoldHill</td>
<td>25.017</td>
<td>0.156</td>
<td>1</td>
<td>28.533</td>
<td>6.9</td>
<td>28.757</td>
<td>19.8</td>
<td>28.792</td>
<td>39.1</td>
<td>65.1</td>
<td>98.6</td>
</tr>
</tbody>
</table>

Average PSNR (dB): 25.181
Average time (sec.): 0.136
Average count: 1
Average ratio (%): 86.43
<table>
<thead>
<tr>
<th>Images</th>
<th>Methods</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FSVQ</td>
<td>30.28</td>
<td>29.55</td>
<td>29.28</td>
</tr>
<tr>
<td>Lena</td>
<td>SCS</td>
<td>0.65</td>
<td>0.71</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>HOSM</td>
<td>1.29</td>
<td>1.32</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td>PVDS(2)</td>
<td>29.31</td>
<td>29.36</td>
<td>29.48</td>
</tr>
<tr>
<td>Time</td>
<td>4.953</td>
<td>0.649</td>
<td>1.029</td>
<td>1.379</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>17.4</td>
<td>31.3</td>
<td>26.4</td>
</tr>
<tr>
<td></td>
<td>28.194</td>
<td>28.194</td>
<td>28.194</td>
<td>28.194</td>
</tr>
<tr>
<td></td>
<td>30.163</td>
<td>30.163</td>
<td>30.163</td>
<td>30.163</td>
</tr>
<tr>
<td>Boat</td>
<td>Time (sec.)</td>
<td>4.969</td>
<td>0.629</td>
<td>1.078</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>13.2</td>
<td>27.7</td>
<td>24.4</td>
</tr>
<tr>
<td></td>
<td>30.517</td>
<td>30.517</td>
<td>30.517</td>
<td>30.517</td>
</tr>
<tr>
<td></td>
<td>30.163</td>
<td>30.163</td>
<td>30.163</td>
<td>30.163</td>
</tr>
<tr>
<td>Jet</td>
<td>Time (sec.)</td>
<td>4.969</td>
<td>0.597</td>
<td>0.859</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>9.6</td>
<td>21.9</td>
<td>22.5</td>
</tr>
<tr>
<td></td>
<td>30.163</td>
<td>30.163</td>
<td>30.163</td>
<td>30.163</td>
</tr>
<tr>
<td>Pepper</td>
<td>Time (sec.)</td>
<td>4.938</td>
<td>0.638</td>
<td>1.141</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>15.5</td>
<td>29.6</td>
<td>24.8</td>
</tr>
<tr>
<td></td>
<td>30.712</td>
<td>30.596</td>
<td>30.707</td>
<td>30.708</td>
</tr>
<tr>
<td></td>
<td>31.408</td>
<td>31.357</td>
<td>31.384</td>
<td>31.383</td>
</tr>
<tr>
<td>Baboon</td>
<td>Time (sec.)</td>
<td>4.953</td>
<td>0.651</td>
<td>1.281</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>17.2</td>
<td>33.0</td>
<td>29.7</td>
</tr>
<tr>
<td></td>
<td>28.503</td>
<td>28.770</td>
<td>28.793</td>
<td>28.798</td>
</tr>
<tr>
<td></td>
<td>29.632</td>
<td>29.614</td>
<td>29.609</td>
<td>29.611</td>
</tr>
<tr>
<td>GoldHill</td>
<td>Time (sec.)</td>
<td>4.969</td>
<td>0.646</td>
<td>1.203</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>16.9</td>
<td>31.3</td>
<td>26.2</td>
</tr>
<tr>
<td></td>
<td>29.133</td>
<td>29.097</td>
<td>29.124</td>
<td>29.127</td>
</tr>
<tr>
<td></td>
<td>29.911</td>
<td>29.892</td>
<td>29.893</td>
<td>29.895</td>
</tr>
<tr>
<td>Average PSNR</td>
<td>27.851</td>
<td>27.821</td>
<td>27.847</td>
<td>27.849</td>
</tr>
<tr>
<td></td>
<td>Average time (sec.)</td>
<td>4.959</td>
<td>0.635</td>
<td>1.128</td>
</tr>
<tr>
<td></td>
<td>Average count</td>
<td>128</td>
<td>15.0</td>
<td>29.1</td>
</tr>
<tr>
<td></td>
<td>Average ratio (%)</td>
<td>100</td>
<td>99.89</td>
<td>99.98</td>
</tr>
<tr>
<td></td>
<td>Duplicated space</td>
<td>377</td>
<td>564</td>
<td>202</td>
</tr>
</tbody>
</table>

**TABLE III**

**Performance Comparison on the Encoding Quality and the Computation Time at Different Codebook Sizes (With Duplication)**
Fig. 6. Comparison results on image quality and computation time among schemes without/with duplication ability at different codebook sizes. (a) Average image quality; (b) average computation time.
\[
1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n!
\]
CONCLUSION
Conclusion

• After two rounds of ripple enlargement, up to 99.69% of the image quality.

• With the help of the duplication mechanism, PVDS can 99.98% of the image quality.

• PVDS is superior to other partial-search VQ methods in terms of both image quality and computation cost.
FUTURE WORKS

• Papers review
• Binary searching methods (TSVQ, LBG and others).
REFERENCES


Thank You for Your Attention !