CSEP 590
Data Compression
Autumn 2007

Scalar Quantization
Vector Quantization
Lossy Image Compression Methods

• DCT Compression
  – JPEG
• Scalar quantization (SQ).
• Vector quantization (VQ).
• Wavelet Compression
  – SPIHT
  – GTW
  – EBCOT
  – JPEG 2000
Scalar Quantization

Source image

Codebook

Index of a codeword

Decoded image
Scalar Quantization Strategies

- Build a codebook with a training set. Encode and decode with fixed codebook.
  - Most common use of quantization
- Build a codebook for each image. Transmit the codebook with the image.
- Training can be slow.
Distortion

• Let the image be pixels $x_1, x_2, \ldots, x_T$.
• Define $\text{index}(x)$ to be the index transmitted on input $x$.
• Define $c(j)$ to be the codeword indexed by $j$.

$$D = \sum_{i=1}^{T} (x_i - c(\text{index}(x_i)))^2 \quad \text{(Distortion)}$$

$$\text{MSE} = \frac{D}{T}$$
Uniform Quantization Example

- 512 x 512 image with 8 bits per pixel.
- 8 codewords

Index | Codeword
-----|---------
0    | 16      
1    | 47      
2    | 79      
3    | 111     
4    | 143     
5    | 175     
6    | 207     
7    | 239     

boundary codeword
Uniform Quantization Example

Encoder

<table>
<thead>
<tr>
<th>input code</th>
<th>0-31</th>
<th>32-63</th>
<th>64-95</th>
<th>96-127</th>
<th>128-159</th>
<th>160-191</th>
<th>192-223</th>
<th>224-255</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
</tbody>
</table>

Decoder

<table>
<thead>
<tr>
<th>code output</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>47</td>
<td>79</td>
<td>111</td>
<td>143</td>
<td>175</td>
<td>207</td>
<td>239</td>
</tr>
</tbody>
</table>

Bit rate = 3 bits per pixel
Compression ratio = 8/3 = 2.67
Example

- $[0, 100)$ with 5 symbols
- $Q = 20$

Encode: $0 = \left\lfloor \frac{10}{20} \right\rfloor$, $1 = \left\lfloor \frac{30}{20} \right\rfloor$, $2 = \left\lfloor \frac{50}{20} \right\rfloor$, ...

Decode: $(0 + \frac{1}{2}) \cdot 20 = 10$, $(1 + \frac{1}{2}) \cdot 20 = 30$, $(2 + \frac{1}{2}) \cdot 20 = 50$, ...
Alternative Uniform Quantization Calculation with Push to Zero

- Range = [min, max)
- Target is S symbols
- Choose Q = (max − min)/S

- Encode x
  \[ s = \left\lfloor \frac{x}{Q} + 1/2 \right\rfloor \]

- Decode s
  \[ x' = s \cdot Q \]
Example

- $[0,90)$ with 5 symbols
- $Q = 20$

<table>
<thead>
<tr>
<th>Encode</th>
<th>Decode</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 = \left[ \frac{9.99}{20} + 1/2 \right]$</td>
<td>$0 \cdot 20 = 0$</td>
</tr>
<tr>
<td>$1 = \left[ \frac{29.99}{20} + 1/2 \right]$</td>
<td>$1 \cdot 20 = 20$</td>
</tr>
<tr>
<td>$2 = \left[ \frac{49.99}{20} + 1/2 \right]$</td>
<td>$2 \cdot 20 = 40$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Improving Bit Rate

Frequency of pixel values

\[ q_j = \text{the probability that a pixel is coded to index } j \]

Potential average bit rate is entropy.

\[ H = \sum_{j=0}^{7} q_j \log_2 \left( \frac{1}{q_j} \right) \]
Example

• 512 x 512 image = 216,144 pixels

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>input frequency</td>
<td>0-31</td>
<td>32-63</td>
<td>64-95</td>
<td>96-127</td>
<td>128-159</td>
<td>160-191</td>
<td>192-223</td>
<td>224-255</td>
</tr>
<tr>
<td>25,000</td>
<td>100,000</td>
<td>90,000</td>
<td>10,000</td>
<td>10,000</td>
<td>10,000</td>
<td>18,000</td>
<td>9,144</td>
<td></td>
</tr>
</tbody>
</table>

Huffman Tree

ABR = (100000 x 1 + 90000 x 2 + 43000 x 4 + 39144 x 5)/216144

= 2.997

Arithmetic coding should work better.
Improving Distortion

• Choose the codeword as a weighted average

\[ c(j) = \text{round}(\sum_{L_j \leq x < R_j} x \cdot p_x) \]

Let \( p_x \) be the probability that a pixel has value \( x \).

Let \([L_j, R_j]\) be the input interval for index \( j \).

\( c(j) \) is the codeword indexed \( j \).
Example

All pixels have the same index.

\[ \begin{array}{cccccccc}
\text{pixel value} & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\text{frequency} & 100 & 100 & 100 & 40 & 30 & 20 & 10 & 0 \\
\end{array} \]

New Codeword = \( \text{round}\left(\frac{8 \cdot 100 + 9 \cdot 100 + 10 \cdot 100 + 11 \cdot 40 + 12 \cdot 30 + 13 \cdot 20 + 14 \cdot 10 + 15 \cdot 0}{400}\right) = 10 \)

Old Codeword = 11

New Distortion = \(140 \cdot 1^2 + 130 \cdot 2^2 + 20 \cdot 3^2 + 10 \cdot 4^2 = 10000\)

Old Distortion = \(130 \cdot 1^2 + 120 \cdot 2^2 + 110 \cdot 3^2 = 16000\)
An Extreme Case

Frequency of pixel values

Only two codewords are ever used!!
Non-uniform Scalar Quantization

Frequency of pixel values

- codeword
  - boundary between codewords
Lloyd Algorithm

- Lloyd (1957)
- Creates an optimized codebook of size n.
- Let $p_x$ be the probability of pixel value $x$.
  - Probabilities might come from a training set
- Given codewords $c(0), c(1), ..., c(n-1)$ and pixel $x$ let $\text{index}(x)$ be the index of the closest code word to $x$.
- Expected distortion is
  \[ D = \sum_x p_x (x - c(\text{index}(x)))^2 \]
- Goal of the Lloyd algorithm is to find the codewords that minimize distortion.
- Lloyd finds a local minimum by an iteration process.
Lloyd Algorithm

Choose a small error tolerance $\varepsilon > 0$.
Choose start codewords $c(0), c(1), \ldots, c(n-1)$
Compute $X(j) := \{x : x \text{ is a pixel value closest to } c(j)\}$
Compute distortion $D$ for $c(0), c(1), \ldots, c(n-1)$
Repeat
  Compute new codewords
    $$c'(j) := \text{round} \left( \sum_{x \in X(j)} x \cdot p_x / p_{X(j)} \right)$$
    Compute $X'(j) = \{x : x \text{ is a pixel value closest to } c'(j)\}$
    Compute distortion $D'$ for $c'(0), c'(1), \ldots, c'(n-1)$
    if $|(D - D')/D| < \varepsilon$ then quit
    else $c := c'; X := X', D := D'$
End{repeat}
Example

Initially c(0) = 2 and c(1) = 5

<table>
<thead>
<tr>
<th>pixel value</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

X(0) = [0,3], X(1) = [4,7]

D(0) = 140 \cdot 1^2 + 100 \cdot 2^2 = 540; D(1) = 40 \cdot 1^2 = 40

D = D(0) + D(1) = 580

c'(0) = \text{round}((100 \cdot 0 + 100 \cdot 1 + 100 \cdot 2 + 40 \cdot 3)/340) = 1

c'(1) = \text{round}((30 \cdot 4 + 20 \cdot 5 + 10 \cdot 6 + 0 \cdot 7)/60) = 5
Example

\[ \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
100 & 100 & 100 & 40 & 30 & 20 & 10 & 0 \\
\end{array} \]

\[ c'(0) = 1; c'(1) = 5 \]
\[ X'(0) = [0,2]; X'(1) = [3,7] \]
\[ D'(0) = 200 \cdot 1^2 = 200 \]
\[ D'(1) = 40 \cdot 1^2 + 40 \cdot 2^2 = 200 \]
\[ D' = D'(0) + D'(1) = 400 \]
\[ \left| (D - D')/D \right| = (580 - 400)/580 = .31 \]
\[ c := c'; X := X'; D := D' \]
Example

\begin{align*}
c(0) &= 1; 
c(1) &= 5 \\
x(0) &= [0,2]; 
x(1) &= [3,7] \\
d &= 400 \\
c'(0) &= \text{round}\left(\frac{(100 \cdot 0 + 100 \cdot 1 + 100 \cdot 2)/300}{100}\right) = 1 \\
c'(1) &= \text{round}\left(\frac{(40 \cdot 3 + 30 \cdot 4 + 20 \cdot 5 + 10 \cdot 6 + 0 \cdot 7)/100}{100}\right) = 4
\end{align*}
Example

<table>
<thead>
<tr>
<th>pixel value</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

c'(0) = 1; c'(1) = 4
X'(0) = [0,2]; X'(1) = [3,7]
D'(0) = 200 \cdot 1^2 = 200
D'(1) = 60 \cdot 1^2 + 10 \cdot 2^2 = 100
D' = D'(0) + D'(1) = 300
|D − D'|/D = (400 − 300)/580 = .17
c := c'; X := X'; D := D'
Example

<table>
<thead>
<tr>
<th>pixel value</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ c(0) = 1; c(1) = 4 \]
\[ X(0) = [0,2]; X(1) = [3,7] \]
\[ D = 400 \]
\[ c'(0) = \text{round} \left( \frac{(100 \cdot 0 + 100 \cdot 1 + 100 \cdot 2)/300}{100} \right) = 1 \]
\[ c'(1) = \text{round} \left( \frac{(40 \cdot 3 + 30 \cdot 4 + 20 \cdot 5 + 10 \cdot 6 + 0 \cdot 7)/100}{100} \right) = 4 \]
Example

c'(0) = 1; c'(1) = 4

X'(0) = [0,2]; X'(1) = [3,7]

D'(0) = 200 \cdot 1^2 = 200

D'(1) = 60 \cdot 1^2 + 10 \cdot 2^2 = 100

D' = D'(0) + D'(1) = 300

\|D - D'\|/D = (300 - 300)/580 = 0

Exit with codeword c(0) = 1 and c(1) = 4.
Scalar Quantization Notes

• Useful for analog to digital conversion.
• Useful for estimating a large set of values with a small set of values.
• With entropy coding yields good lossy compression.
• Lloyd algorithm works very well in practice, but can take many iterations.
  – For n codewords should use about 20n size representative training set.
  – imagine 1024 codewords.
Vector Quantization

source image

codebook

index of nearest codeword

i

decoded image
Vectors

• An a x b block can be considered to be a vector of dimension ab.

\[
\begin{array}{c|c|c|c}
w & x \\
\hline
y & z \\
\end{array}
\quad = \quad (w,x,y,z) \text{ vector}
\]

• Nearest means in terms of Euclidian distance or Euclidian squared distance. Both equivalent.

\[
\text{Distance} = \sqrt{(w_1 - w_2)^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}
\]

Squared Distance = \( (w_1 - w_2)^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 \)

• Squared distance is easier to calculate.
Vector Quantization Facts

• The image is partitioned into a x b blocks.
• The codebook has n representative a x b blocks called codewords, each with an index.
• Compression with fixed length codes is

\[
\frac{\log_2 n}{ab} \text{ bpp}
\]

• Example: a = b = 4 and n = 1,024
  – compression is 10/16 = .63 bpp
  – compression ratio is 8 : .63 = 12.8 : 1
• Better compression with entropy coding of indices
Examples

4 x 4 blocks
.63 bpp

4 x 8 blocks
.31 bpp

8 x 8 blocks
.16 bpp

Codebook size = 1,024
Scalar vs. Vector

• Pixels within a block are correlated.
  – This tends to minimize the number of codewords needed to represent the vectors well.

• More flexibility.
  – Different size blocks
  – Different size codebooks
Encoding and Decoding

• Encoding:
  – Scan the a x b blocks of the image. For each block find the nearest codeword in the codebook and output its index.
  – Nearest neighbor search.

• Decoding:
  – For each index output the codeword with that index into the destination image.
  – Table lookup.
The Codebook

- Both encoder and decoder must have the same codebook.
- The codebook must be useful for many images and be stored someplace.
- The codebook must be designed properly to be effective.
- Design requires a representative training set.
- These are major drawbacks to VQ.
Codebook Design Problem

- Input: A training set \( X \) of vectors of dimension \( d \) and a number \( n \). (\( d = a \times b \) and \( n \) is number of codewords)

- Output: \( n \) codewords \( c(0), c(1), \ldots, c(n-1) \) that minimize the distortion.

\[
D = \sum_{x \in X} \| x - c(\text{index}(x)) \|^2
\]

where \( \text{index}(x) \) is the index of the nearest codeword to \( x \).

\[
\|(x_0, x_1, \ldots, x_{d-1})\|^2 = x_0^2 + x_1^2 + \cdots + x_{d-1}^2
\]

squared norm
GLA

- The Generalized Lloyd Algorithm (GLA) extends the Lloyd algorithm for scalars.
  - Also called LBG after inventors Linde, Buzo, Gray (1980)
- It can be very slow for large training sets.
GLA

Choose a training set \( X \) and small error tolerance \( \varepsilon > 0 \).
Choose start codewords \( c(0), c(1), \ldots, c(n-1) \)
Compute \( X(j) := \{x : x \text{ is a vector in } X \text{ closest to } c(j)\} \)
Compute distortion \( D \) for \( c(0), c(1), \ldots, c(n-1) \)
Repeat
  Compute new codewords
    \[ c'(j) := \text{round} \left( \frac{1}{|X(j)|} \sum_{x \in X(j)} x \right) \] (centroid)
  Compute \( X'(j) = \{x : x \text{ is a vector in } X \text{ closest to } c'(j)\} \)
  Compute distortion \( D' \) for \( c'(0), c'(1), \ldots, c'(n-1) \)
  if \( |(D - D')/D| < \varepsilon \) then quit
  else \( c := c'; X := X', D := D' \)
End{repeat}
GLA Example (1)

- Codeword
- Training vector

- $c(0)$
- $c(1)$
- $c(2)$
- $c(3)$
GLA Example (2)

- **Codeword**
- **Training Vector**

Diagram:
- $X(0)$
- $X(1)$
- $X(2)$
- $X(3)$
GLA Example (3)

X(0)

- codeword
- training vector
- centroid

\( c(0) \)
\( c'(0) \)
GLA Example (4)
GLA Example (5)
GLA Example (6)
GLA Example (7)
GLA Example (8)

codeword

training vector
GLA Example (9)

codeword

training vector

centroid
GLA Example (10)
1 x 2 codewords

Note: codewords diagonally spread
Codeword Splitting

• It is possible that a chosen codeword represents no training vectors, that is, $X(j)$ is empty.
  - *Splitting* is an alternative codebook design algorithm that avoids this problem.

• Basic Idea
  - Select codeword $c(j)$ with the greatest distortion.
    \[ D(j) = \sum_{x \in X(j)} \| x - c(j) \|^2 \]
  - Split it into two codewords then do the GLA.
Example of Splitting

Initially $c(0)$ is centroid of training set
Example of Splitting

Split $c(1) = c(0) + \varepsilon$
Example of Splitting

- codeword
- training vector
- Apply GLA

- c(0)
- c(1)
Example of Splitting

codeword

training vector

c(0) has max distortion so split it.
Example of Splitting

codeword

training vector

Apply GLA
Example of Splitting

codeword

training vector

c(2) has max distortion so split it

X(0)

X(1)

X(2)
Example of Splitting
GLA Advice

• Time per iteration is dominated by the partitioning step, which is $m$ nearest neighbor searches where $m$ is the training set size.
  – Average time per iteration $O(m \log n)$ assuming $d$ is small.

• Training set size.
  – Training set should be at least 20 training vectors per code word to get reasonable performance.
  – Too small a training set results in “over training”.

• Number of iterations can be large.
Encoding

• Naive method.
  – For each input block, search the entire codebook to find the closest codeword.
  – Time $O(T \cdot n)$ where $n$ is the size of the codebook and $T$ is the number of blocks in the image.
  – Example: $n = 1024$, $T = 256 \times 256 = 65,536$ (2 x 2 blocks for a 512 x 512 image)
    $nT = 1024 \times 65536 = 2^{26} \approx 67$ million distance calculations.

• Faster methods are known for doing “Full Search VQ”. For example, k-d trees.
  – Time $O(T \log n)$
VQ Encoding is Nearest Neighbor Search

• Given an input vector, find the closest codeword in the codebook and output its index.
• Closest is measured in squared Euclidian distance.
• For two vectors \((w_1,x_1,y_1,z_1)\) and \((w_2,x_2,y_2,z_2)\).

\[
\text{Squared Distance} = (w_1 - w_2)^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2
\]
k-d Tree

• Jon Bentley, 1975
• Tree used to store spatial data.
  – Nearest neighbor search.
  – Range queries.
  – Fast look-up
• k-d tree are guaranteed $\log_2 n$ depth where $n$ is the number of points in the set.
  – Traditionally, k-d trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.
k-d Tree Construction

- If there is just one point, form a leaf with that point.
- Otherwise, divide the points in half by a line perpendicular to one of the axes.
- Recursively construct k-d trees for the two sets of points.
- Division strategies
  - divide points perpendicular to the axis with widest spread.
  - divide in a round-robin fashion.
k-d Tree Construction (1)

divide perpendicular to the widest spread.
k-d Tree Construction (2)
k-d Tree Construction (3)
k-d Tree Construction (4)
k-d Tree Construction (5)
k-d Tree Construction (6)
k-d Tree Construction (7)
k-d Tree Construction (8)
k-d Tree Construction (9)
k-d Tree Construction (10)
k-d Tree Construction (11)
k-d Tree Construction (12)
k-d Tree Construction (13)
k-d Tree Construction (14)
k-d Tree Construction (15)
k-d Tree Construction (16)
k-d Tree Construction (17)
k-d Tree Construction (18)
k-d Tree Construction Complexity

• First sort the points in each dimension.
  – O(dn log n) time and dn storage.
  – These are stored in A[1..d,1..n]
• Finding the widest spread and equally dividing into two subsets can be done in O(dn) time.
• Constructing the k-d tree can be done in O(dn log n) and dn storage
k-d Tree Codebook Organization

2-d vectors
(x,y)

CSEP 590 - Lecture 8 - Autumn 2007
Node Structure for k-d Trees

- A node has 5 fields
  - axis (splitting axis)
  - value (splitting value)
  - left (left subtree)
  - right (right subtree)
  - point (holds a point if left and right children are null)
k-d Tree Nearest Neighbor Search

NNS(q: point, n: node, p: ref point w: ref distance)
if n.left = n.right = null then {leaf case}
   w' := ||q - n.point||;
   if w' < w then w := w'; p := n.point;
else
   if w = infinity then
      if q(n.axis) ≤ n.value then
         NNS(q, n.left, p, w);
         if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
      else
         NNS(q, n.right, p, w);
         if q(n.axis) - w < n.value then NNS(q, n.left, p, w)
   else {w is finite}
      if q(n.axis) - w ≤ n.value then NNS(q, n.left, p, w)
      if q(n.axis) + w > n.value then NNS(q, n.right, p, w);

initial call NNS(q, root, p, infinity)
Explanation

q(n.axis) - w ≤ n.value means the circle overlaps the left subtree.

q(n.axis) + w > n.value means the circle overlaps the right subtree.
k-d Tree NNS (1)

query point

CSEP 590 - Lecture 8 - Autumn 2007
k-d Tree NNS (2)

query point

CSEP 590 - Lecture 8 - Autumn 2007
k-d Tree NNS (3)

query point
k-d Tree NNS (4)

query point

- a, b, c, d, e, x, y, i

- s1, s2, s3, s4, s5, s6, s7, s8
k-d Tree NNS (5)

query point

CSEP 590 - Lecture 8 - Autumn 2007
k-d Tree NNS (6)

query point

CSEP 590 - Lecture 8 - Autumn 2007
k-d Tree NNS (7)

query point

CSEP 590 - Lecture 8  - Autumn 2007
k-d Tree NNS (8)

query point

CSEP 590 - Lecture 8 - Autumn 2007
k-d Tree NNS (9)

query point

CSEP 590 - Lecture 8 - Autumn 2007
k-d Tree NNS (10)

query point

CSEP 590 - Lecture 8  - Autumn 2007
k-d Tree NNS (11)
k-d Tree NNS (12)

query point
k-d Tree NNS (13)
k-d Tree NNS (14)

query point
k-d Tree NNS (15)
k-d Tree NNS (16)
k-d Tree NNS (17)

- Query point

Diagram showing a k-d tree with labeled nodes and data points, illustrating the search process in a k-d tree.
k-d Tree NNS (18)

query point

CSEP 590 - Lecture 8 - Autumn 2007
k-d Tree NNS (19)

query point

CSEP 590 - Lecture 8 - Autumn 2007
k-d Tree NNS (20)

query point

CSEP 590 - Lecture 8 - Autumn 2007
k-d Tree NNS (21)

query point

CSEP 590 - Lecture 8 - Autumn 2007
Notes on k-d Tree NNS

- Has been shown to run in $O(\log n)$ average time per search in a reasonable model. (Assume $d$ a constant)
- For VQ it appears that $O(\log n)$ is correct.
- Storage for the k-d tree is $O(n)$.
- Preprocessing time is $O(n \log n)$ assuming $d$ is a constant.
Alternatives

• Orchard’s Algorithm (1991)
  – Uses $O(n^2)$ storage but is very fast

• Annulus Algorithm
  – Similar to Orchard but uses $O(n)$ storage. Does many more distance calculations.

• PCP Principal Component Partitioning
  – Zatloukal, Johnson, Ladner (1999)
  – Similar to k-d trees
  – Also very fast
Principal Component Partition
PCP Tree vs. k-d tree
Comparison in Time per Search

4,096 codewords
Notes on VQ

• Works well in some applications.
  – Requires training
• Has some interesting algorithms.
  – Codebook design
  – Nearest neighbor search
• Variable length codes for VQ.
  – PTSVQ - pruned tree structured VQ (Chou, Lookabaugh and Gray, 1989)
  – ECVQ - entropy constrained VQ (Chou, Lookabaugh and Gray, 1989)