Fractal Compression

- Founders: Manderbrot and Barnsley.

- Basic idea: fixed point transformation

- $X_0$ is fixed point of function $f$ if
  \[ f(X_0) = X_0 \]

- Example: Transformation $ax + b$
  has a fixed point $X_0$ given by:
  \[ X_0 = aX_0 + b \]

- To transmit $X_0$, send $a, b$

Then iterate:

\[ X_0^{(n+1)} = aX_0^{(n)} + b \]

will converge regardless of initial guess.
Image Compression

• Think of Image I as array of numbers

• Find a function \( f \) such that

\[
f(I) = I
\]

• If # of bits to transmit \( f \) is smaller than I, achieve Compression

• In practice, hard to come up with one transformation \( f \), for the whole image.

• Divide up the image into domain and domain and Range blocks
Image Compression

- Main idea:
  - Divide up image into $M \times M$ "Range" blocks
  - For each Range block find another $2M \times 2M$ "Domain block" from the same image such that for some transformation $f_K$ we get $f_K(D_K) = R_K$
    
    $D_K = \text{Domain block } k$
    $R_K = \text{Range block } k$

- First publicly discussed by Jacquin in 1989 thesis + 1992 paper

- Works well if there is self similarity in image.
• What should $f_K$ do?
  - change **size** of domain block
  - change **orientation** of domain block
  - change **intensity** of pixels

• $f_K$ consists of
  - geometric transformation : $g_K$
  - massic transformation : $m_K$

• $g_K$ : displacement + size + **intensity**

• $m_K$ : orientation + **intensity**
Transformations:

- $g_K$: displaces + adjusts intensity
  \[ \overset{\longrightarrow}{\text{easy}} \]

- $m_K$: $m_K(t_{ij}) = i(\alpha_K t_{ij} + \Delta_K)$
  
  $i$ can be
  
  * Rotation by 90, 180, -90
  * Reflection about
    horizontal, vertical, diagonal
  * identity map

- Finding transformations is compute intensive

- Search through all domain blocks + all transformations to find "BEST" one

- Encoding more time than decoding
• If Image is divided \( m_{t_0} \)

\[ N \text{ Range blocks} \rightarrow N \text{ transformations} \quad f_K \quad k = 1, \ldots, N \]

are its representation.

\[ f = \bigcup_k f_k \]

\[ \hat{I} = f(\hat{I}) \]

\( \hat{I} \) is approximation to \( I \).

• Collage theorem guarantees convergence to \( \hat{I} \) using any arbitrary initial guess for image.
FIGURE 13.11 The first six iterations of the fractal decoding process.
VECTOR QUANTIZATION

- Let $\vec{f}$ denote $N$ dimensional vectors consisting of $N$ real valued, continuous amplitude scalars.
- **Basic Idea:** Map $\vec{f}$ into $L$ possible $N$ dimensional reconstruction vectors $\vec{r}_i$ for $1 \leq i \leq L$.

$$\hat{\vec{f}} = VQ(\vec{f}) = \vec{r}_i \quad \vec{f} \in C_i$$

- Define a distortion measure:

$$D = E[(\hat{\vec{f}} - \vec{f})^T(\hat{\vec{f}} - \vec{f})]$$

$$= \sum_{i=1}^{L} \int_{\vec{f}_0 \in C_i} (\vec{r}_i - \hat{\vec{f}}_0) d\vec{f}_0$$

![Diagram](image)

*Figure 10.8 Example of vector quantization. The number of scalars in the vector is 2, and the number of reconstruction levels is 9.*
Properties of Vector Quantization

- Removes linear dependency between random variables.
- Removes nonlinear dependency between random variables.
- Explits increase in dimensionality.
- Allows us to code a scalar with less than one bit.
- Computational and storage requirements are far greater than scalar quantization.
VQ Removes Linear Dependency

- Linear transformation can decorrelate linearly dependent (correlated) random variables.

![Diagram](image)

Figure 10.9 Illustration that vector quantization can exploit linear dependence of scalars in the vector. (a) Probability density function $p_{f_1 f_2}(f_1, f_2)$, (b) reconstruction levels (filled-in dots) in scalar quantization; (c) reconstruction levels (filled-in dots) in vector quantization.

![Diagram](image)

Some distortion but reduce # of reconstruction levels.

Rotate $f_1, f_2$ to get $g_1, g_2$ uncorrelated.

Figure 10.10 Result of eliminating linear dependence of the two scalars $f_1$ and $f_2$ in Figure 10.9 by linear transformation of $f_1$ and $f_2$.
VQ Removes Nonlinear Dependency

- Nonlinear dependence cannot be eliminated by a linear operator.

Figure 10.11 Illustration that vector quantization can exploit nonlinear dependence of scalars in the vector. (a) Probability density function $p_{u,v}(u, v)$; (b) reconstruction levels (solid dots) in scalar quantization; (c) reconstruction levels (solid dots) in vector quantization.
VQ Exploits the Increase in Dimensionality

- The mean square error due to VQ is approximately less than 4 percent than scalar quantization with same # of reconstruction levels.

![Diagram](image)

**Figure 10.13** Illustration that vector quantization can exploit the dimensionality increase. In this case, the mean square error due to vector quantization is approximately 4% less than that due to scalar quantization. (a) Scalar quantization of $f_1$ and $f_2$; (b) vector quantization of $f_1$ and $f_2$. 

$\Rightarrow$ # of bits per scalar with VQ $\leq 1$

Look at 10.9 scalar 1 bit per scalar

VQ: $\frac{1}{n}$ bit per scalar
Codebook Design Algorithms

- K-means algorithm.
- Tree codebooks and binary search.
- Nearest neighbor.
Codebook Design via K-means

- Exploit the following two necessary conditions for the optimal solution:
  - For a vector $\vec{f}$ to be quantized to one of the reconstruction levels, the optimal quantizer must choose the reconstruction level $\vec{r}_i$ which has the smallest distortion between $\vec{f}$ and $\vec{r}_i$: $\exists \vec{r}_i \quad d(\vec{f}, \vec{r}_i) \leq d(\vec{f}, \vec{r}_j) \quad j \neq i$
  - Each reconstruction level $\vec{r}_i$ must minimize the average distortion $D$ in $C_i$.
    
    Minimize $E[d(\vec{f}, \vec{r}_i) \mid \vec{f} \in C_i]$ w.r.t. $\vec{r}_i$

- Find $\vec{r}_i$ and $C_i$ iteratively $\rightarrow$ Problem: local versus global minimum $\rightarrow$ initial guess important.

![Flowchart of K-means codebook design](image)

Figure 10.14 Codebook design by the K-means algorithm for vector quantization.
Complexity of K-means

- M training vectors, L codewords, N dimensional, R bits per scalar.

- Complexity of Codebook design:
  - $ML$ evaluation of distortion measure for each iteration.
  - $MLN = NM2^{NR}$ additions and mults per iteration.
  - Example: $N = 10$, $R = 2$, $M = 10L$ results in 100 trillion operation per iteration.
  - Storage: MN for training vectors, LN for reconstruction levels $\rightarrow (M + 2^{NR})N$.

- Complexity of operation at the receiver.
  - Storage of reconstruction levels: $N2^{NR}$. If $N = 10$ and $R = 2$, storage is 10 million.
  - Number of arithmetic operations $N2^{NR}$. If $N = 10$ and $R = 2$, 10 million operations per look up.
Tree Codebook and Binary Search

- Full search is responsible for exponential growth of the number of operations at the receiver → Tree codebook.

- Let $L$ be a power of 2.

- Basic operation of tree codebook design:
  - Use K-means to divide the $N$ dimensional space of $f$ into two regions.
  - Divide each of the two regions into two more regions using the K-means algorithm.
  - Repeat step 2 until there are $L$ reconstruction levels.

![Image of a tree codebook](image-url)
$M = \# \text{ of Training Vectors}$
$L = \# \text{ of codebook}$
$N = \text{Dimension}$

- Complexity of Tree Codebook

- Design complexity:
  - Number of arithmetic operations per iteration is $2NM \log_2 L$. For $N = 10$ and $R = 2$, the reduction factor compared to the full search is 26,000.
  - Storage: approximately the same as full search algorithm.

- Operation complexity at receiver:
  - Number of arithmetic operations is $2N^2 R$. For $N = 10$ and $R = 2$, the reduction factor compared to the full search is 26,000.
  - Storage: The codebook must store all the intermediate reconstruction levels as well as the final reconstruction levels. → Twice as much storage needed as full search.

- Distortion of full search is slightly smaller than that of tree search.
Nearest Neighbor Design Algorithm

- Initially proposed by Equitz.
- Computational complexity grows linearly with the training set.
- Find the 2 vectors closest to each other, merge them into another vector equal to their mean, repeat this process until the number of vectors is $L$.
- Main efficiency is achieved by partitioning the training data into a K-D tree $\rightarrow$ multiple merges at each iteration.
Variations of VQ

- Multistage VQ reduces storage and search time.
  1. First stage a low rate VQ.
  2. Generate error by subtracting the codeword from the original.
  3. Code the error by a different VQ.
  4. Repeat steps 2 and 3.

![Diagram of Multistage Vector Quantization](image)

Fig. 5.5.2 Multistage Vector Quantization. At each stage an error vector is computed, which is then used as the input to the next stage of VQ. The decoder merely computes a summation of the code vectors corresponding to the received indices.

- Parameter extraction techniques:
  - mean and variance of each input vector are computed and sent separately.
  - mean and variance might be coded with DPCM.
Variations of VQ (cont’d)

• Block classification:
  – Divide the blocks into several classes according to spatial activity.
  – Design a codebook for each class.
  – Overhead on transmitting the codebook is large.

• Combine prediction techniques with VQ:
  – Coded quantity is the prediction error rather than intensity values.

• VQ of color images exploits the correlation between color components.

• Typical rates: .1 to .5 bits per pixel for $4 \times 4$ pixels as vectors.
local measure, such as local image contrast. If the local measure used can be obtained from previously coded pixel intensities, then it does not have to be transmitted. If the local measure used is obtained directly from \( f(n_1, n_2) \), it must be transmitted, since the receiver does not have access to the original image. Adaptive coding certainly adds complexity to an image coder, but can often significantly improve its performance.

PCM systems do not exploit the statistical dependence of neighborhood pixel intensities. One way to exploit the statistical dependence is to use methods such as DM, DPCM, and two-channel coders, where the difference between \( f(n_1, n_2) \) and a prediction of \( f(n_1, n_2) \) is coded. An alternate way is to use vector quantization. As we discussed in Section 10.2, vector quantization can exploit the statistical dependence of the parameters coded. Vector quantization has been considered in coding the waveform \( f(n_1, n_2) \). The blocks used consist of neighborhood pixel intensities of a small size, typically \( 2 \times 2, 3 \times 3, \) and \( 4 \times 4 \). Vector quantization has been primarily applied in low bit rate (below 1 bit/pixel) applications, since the computational and storage costs increase rapidly with the block size and the bit rate. Intelligible images can be reconstructed with some sacrifice in quality at bit rates below 1 bit/pixel, which is not possible with DM, DPCM, or two-channel coders with scalar quantization and uniform-length codeword assignment. For waveform coders, vector quantization is an effective way to code an image at a bit rate lower than 1 bit/pixel. Figure 10.40 illustrates the performance of vector quantization when applied to coding the waveform \( f(n_1, n_2) \). Figure 10.40(a) shows an original image of \( 512 \times 512 \) pixels. Figure 10.40(b) shows the

![Figure 10.40](image)

**Figure 10.40** Example of an image coded by vector quantization. Courtesy of William Equitz. (a) Original image of \( 512 \times 512 \) pixels; (b) coded image by vector quantization at 1/2 bit/pixel. The block size used is \( 4 \times 4 \) pixels and the codebook is designed by using a variation of the K-means algorithm. NMSE = 2.7%, SNR = 15.7 dB.