7.6 Arithmetic Coding

- Arithmetic coding is a more modern coding method that usually out-performs Huffman coding.
- Huffman coding assigns each symbol a codeword which has an integral bit length. Arithmetic coding can treat the whole message as one unit.
- A message is represented by a half-open interval $[a, b)$ where a and b are real numbers between 0 and 1. Initially, the interval is $[0, 1)$. When the message becomes longer, the length of the interval shortens and the number of bits needed to represent the interval increases.

ALGORITHM 7.5 Arithmetic Coding Encoder

```
BEGIN
   low = 0.0;   high = 1.0;   range = 1.0;

   while (symbol != terminator)
       {
         get (symbol);
         low =  low + range * Range_low(symbol);
         high = low + range * Range_high(symbol);
         range = high - low;
       }

   output a code so that low <= code < high;
END
```

Example: Encoding in Arithmetic Coding

Symbol	Probability	Range
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3, 0.5)
D	0.05	[0.5, 0.55)
E	0.3	[0.55, 0.85)
F	0.05	[0.85, 0.9)
$	0.1	[0.9, 1.0)

(a) Probability distribution of symbols.

Fig. 7.8: Arithmetic Coding: Encode Symbols "CAEE$"

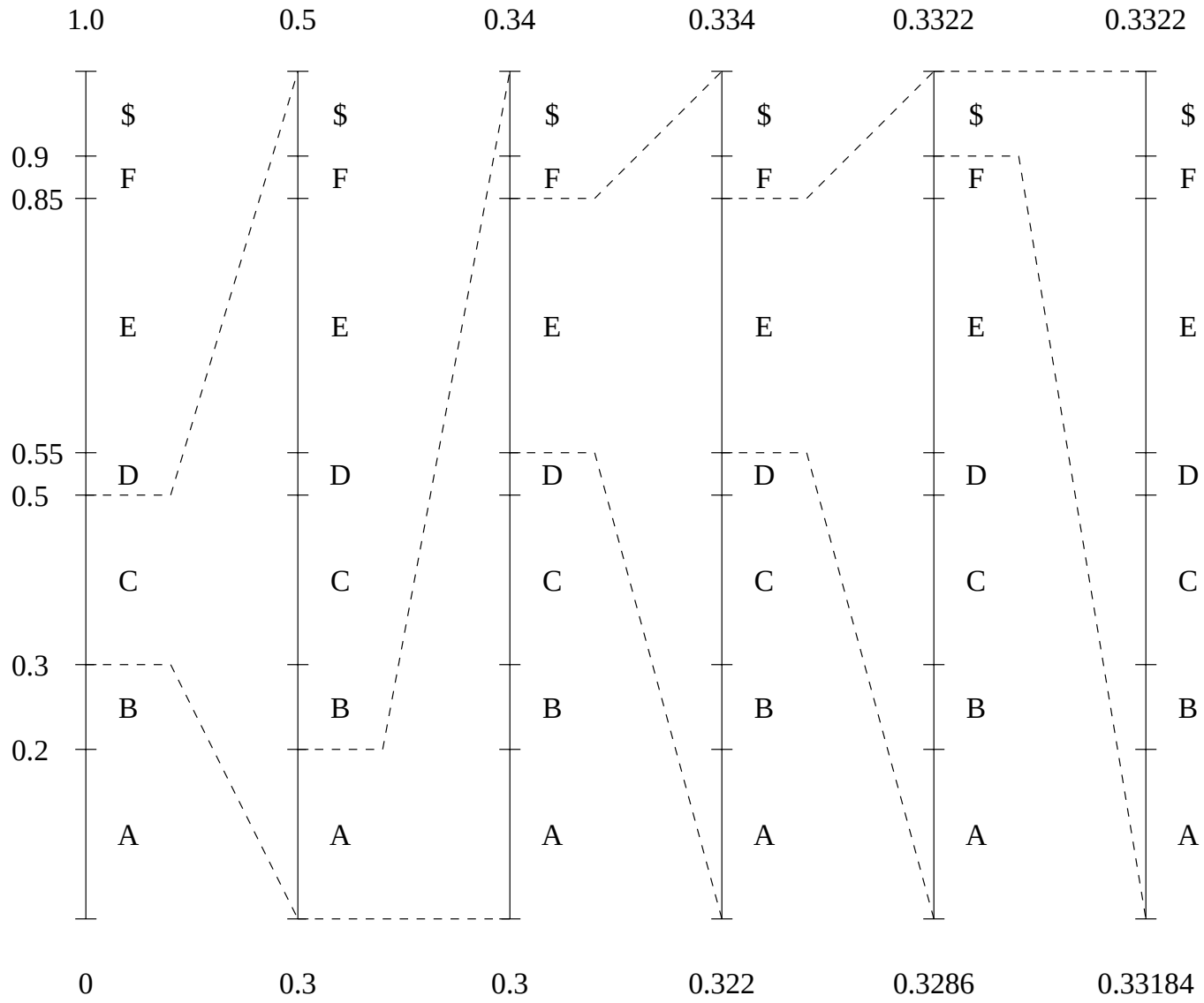

Fig. 7.8(b) Graphical display of shrinking ranges.

Symbol	low	high	range
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
$	0.33184	0.33220	0.00036

(c) New *low*, *high*, and *range* generated.

Fig. 7.8 (cont'd): Arithmetic Coding: Encode Symbols "CAEE$"

PROCEDURE 7.2 Generating Codeword for Encoder

```
BEGIN
   code = 0;
   k = 1;
   while (value(code) < low)
       { assign 1 to the kth binary fraction bit
         if (value(code) > high)
            replace the kth bit by 0

         k = k + 1;
       }
END
```

- The final step in Arithmetic encoding calls for the generation of a number that falls within the range $[low, high)$. The above algorithm will ensure that the shortest binary codeword is found.

ALGORITHM 7.6 Arithmetic Coding Decoder

```
BEGIN
   get binary code and convert to
       decimal value = value(code);
   Do
       { find a symbol s so that
             Range_low(s) <= value < Range_high(s);
         output s;
         low = Rang_low(s);
         high = Range_high(s);
         range = high - low;
         value = [value - low] / range;
       }
   Until symbol s is a terminator
END
```

Table 7.5 Arithmetic coding: decode symbols "CAEE$"

value	Output Symbol	low	high	range
0.33203125	C	0.3	0.5	0.2
0.16015625	A	0.0	0.2	0.2
0.80078125	E	0.55	0.85	0.3
0.8359375	E	0.55	0.85	0.3
0.953125	$	0.9	1.0	0.1

7.7 Lossless Image Compression

- **Approaches of Differential Coding of Images:**
  - Given an original image $I(x, y)$, using a simple difference operator we can define a difference image $d(x, y)$ as follows:

$$d(x, y) = I(x, y) - I(x - 1, y) \tag{7.9}$$

    or use the discrete version of the 2-D Laplacian operator to define a difference image $d(x, y)$ as

$$d(x, y) = 4\,I(x, y) - I(x, y - 1) - I(x, y + 1) - I(x + 1, y) - I(x - 1, y) \tag{7.10}$$

- Due to *spatial redundancy* existed in normal images I, the difference image d will have a narrower histogram and hence a smaller entropy, as shown in Fig. 7.9.

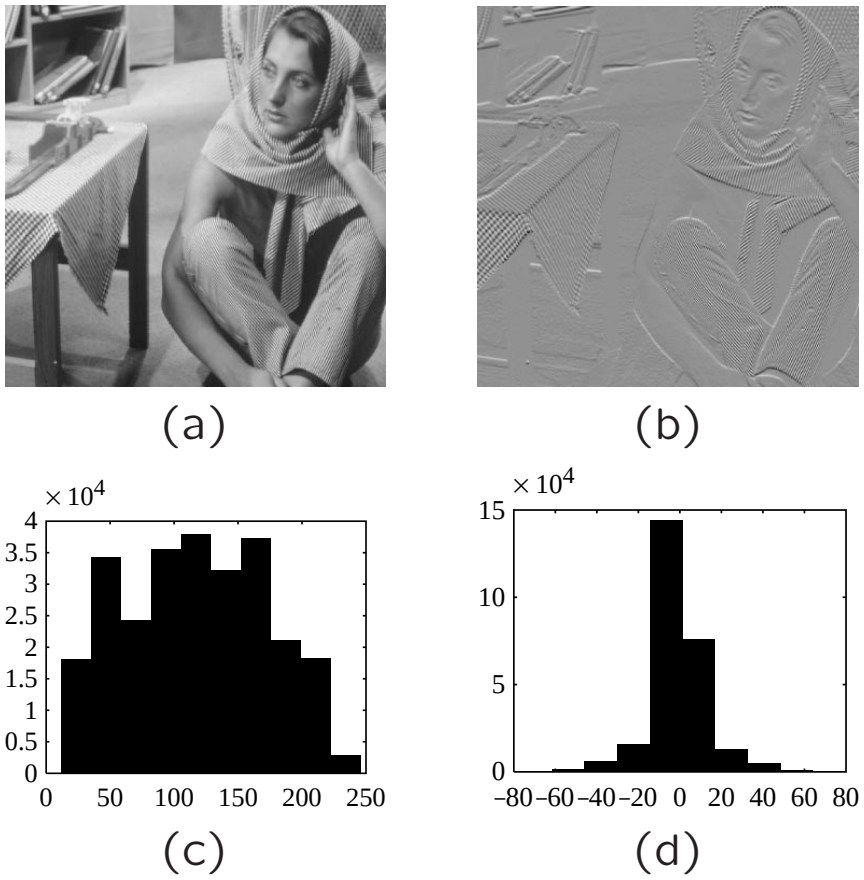

Fig. 7.9: Distributions for Original versus Derivative Images. (a,b): Original gray-level image and its partial derivative image; (c,d): Histograms for original and derivative images.

(This figure uses a commonly employed image called "Barb".)