Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Professor Dr. W. Effelsberg
Christoph Kuhmünch

Multimedia-Technik
WS96/97
Übungsaufgaben

Lösung zu Übungsblatt 4

1 Aufgabe - JPEG

Wir wollen anhand einer Beispielmatrix der Größe 8×8 den JPEG-Algorithmus nachvollziehen. Gegeben sei folgende Matrix:

$$A = \begin{pmatrix} 150 & 170 & 132 & 185 & 147 & 190 & 215 & 220 \\ 165 & 185 & 130 & 190 & 175 & 195 & 223 & 199 \\ 155 & 163 & 180 & 220 & 202 & 173 & 197 & 170 \\ 143 & 154 & 160 & 170 & 211 & 185 & 190 & 166 \\ 130 & 140 & 172 & 190 & 193 & 150 & 180 & 140 \\ 135 & 164 & 198 & 180 & 177 & 141 & 172 & 135 \\ 170 & 190 & 163 & 140 & 165 & 132 & 160 & 140 \\ 160 & 200 & 145 & 135 & 170 & 199 & 190 & 129 \end{pmatrix}$$

1. **DCT**

   (a) Was wird durch die DCT bewirkt?

   - Transformationskodierung: Die Pixelwerte der Matrix werden in den Frequenzraum verschoben.
   - Höhere Frequenzen befinden sich in der rechten unteren Ecke.
   - Da angenommen werden kann, daß ein 8×8 Block eines Bildes relativ gleichmäßig ist, werden viele Vektoren der Frequenzmatrix klein oder gleich $\mathtt{Null}$ sein.

   (b) Welche Eigenschaften des menschlichen Auges werden durch DCT ausgenutzt?

   - Das menschliche Auge nimmt niedrigere Frequenzen besser war als hohe. Mit Hilfe von DCT werden die niedrigen Frequenzen in der rechten unteren Ecke der Matrix konzentriert. Diese können durch höhere Quantisierungsvektoren auf $\mathtt{Null}$ reduziert werden.

   (c) Programmieren Sie die $\mathtt{DCT}$, wie sie in der Vorlesung beschrieben wird. Vergessen Sie dabei nicht, daß die Matrix vor der Transformation in das Intervall $[-128;\ 127]$ verschoben werden muß. Gehen Sie davon aus, daß sich die Ausgangswerte im Intervall $[0;\ 255]$ befinden.

```
typedef int matrix[8][8];
```

```
void dct(matrix& S, matrix& SS)
{
  int    x, y, u, v;
  double Cu, Cv;
  double result;

  for (v= 0; v < 8; v++)
    for (u= 0; u < 8; u++)
      S[v][u]-= 128;

  for (v= 0; v < 8; v++)
    for (u= 0; u < 8; u++)
      {
        Cu= (u == 0) ? 1.0 / sqrt(2.0) : 1.0;
        Cv= (v == 0) ? 1.0 / sqrt(2.0) : 1.0;

        result= 0.0;
        for (x= 0; x < 8; x++)
          for (y= 0; y < 8; y++)
            result+= S[y][x] * cos((2.0*x+1) * u * M_PI / 16.0) *
                                 cos((2.0*y+1) * v * M_PI / 16.0);
        result*= 0.25 * Cu * Cv;
        SS[v][u]=  (int) result;
      }
};


void rdct(matrix& S, matrix& SS)
{
  int    x, y, u, v;
  double Cu, Cv;
  double result;

  for (x= 0; x < 8; x++)
    for (y= 0; y < 8; y++)
      {
        result= 0.0;
        for (u= 0; u < 8; u++)
          for (v= 0; v < 8; v++)
            { Cu= (u == 0) ? 1.0 / sqrt(2.0) : 1.0;
              Cv= (v == 0) ? 1.0 / sqrt(2.0) : 1.0;

              result+= Cu * Cv * S[v][u] * cos((2.0*x+1) * u * M_PI / 16.0) *
                                           cos((2.0*y+1) * v * M_PI / 16.0);
            }
        result*= 0.25;
        SS[y][x]= (int) result +128;
      }
};
```

(d) Berechnen Sie mit Hilfe ihres Programmes die Matrix

$$A' = \mathrm{FDCT}(A)$$

$$A' = \begin{pmatrix} 337 & -48 & -39 & 9 & -22 & 11 & -63 & 5 \\ 62 & -77 & 16 & -12 & 34 & 23 & -5 & -41 \\ 7 & -7 & 75 & 26 & -26 & -12 & -10 & -43 \\ -30 & 1 & 7 & -40 & 5 & 12 & 3 & 3 \\ -9 & -34 & 0 & 24 & -10 & -7 & 7 & -8 \\ -9 & 10 & 14 & -9 & 10 & -12 & 10 & 10 \\ 10 & 2 & -29 & -20 & -7 & 14 & 9 & 0 \\ -8 & 14 & -8 & -16 & 15 & 13 & -2 & 2 \end{pmatrix}$$

2. **Quantifizierung**

   (a) Beschreiben Sie den Quantifizierungsvorgang. Sei B die 8×8 Matrix der Quantifizierungsvektoren. In welchem Segment der Matrix würden Sie tendentiell höhere Quantifizierungsvektoren erwarten?

   - Bei der Quantifizierung wird jeder einzelne Wert der Matrix A' durch einen Quantifizierungskoeffizienten $q_{i,j}$ dividiert. Das Ergebnis der Division wird auf den nächsten ganzzahligen Wert gerundet. Fuer jede Matrixposition gibt es einen eigenen Koeffizienten.
     $$a'_{i,j} = \frac{\mathtt{round}(a'_{i,j})}{q_{i,j}}$$
   - Da für das menschliche Sehvermögen niedrigere Frequenzen tendentiell wichtiger sind, werden die Koeffizienten in der linken oberen Ecke der MAtrix niedriger sein als in der rechten unteren. Da der DC Koeffizient als Grundfarbton der Matrix den höchsten Stellenwert hat, wird dieser den kleinsten Quantifizierungskoeffizienten haben.

   (b) Zur Vereinfachung nehmen wir an, daß die Matrix B nur einen Quantifizierungsvektor ($B = (30)$) enthält. Berechnen Sie

   $A'' = A' \cdot B$

   $$A'' = \begin{pmatrix} 11 & -1 & -1 & 0 & 0 & 0 & -2 & 0 \\ 2 & -2 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

3. **Entropiekodierung**

   (a) Beschreiben Sie die Entropiekodierung.

   - Die Entropiekodierung benutzt Huffmann-Code und Lauflängencodierung zur Abspeicherung der Matrizzen.
   - Der DC-Koeffizient wird als Differenzwert abgespeichert.
   - Die Verarbeitung der restlichen Matrix erfolgt in Zick-Zack Reihenfolge in folgendem Format:

i. Anzahl der folgenden Koeffizienten mit dem Wert Null,
ii. Anzahl der für die Darstellung benötigten Bits,
iii. Wert kodiert mit der angegebenen Anzahl von Bits

(b) Der Standard sieht vor, daß die Werte in Zick-Zack-Linie abzuspeichern sind. Warum ist dies sinnvoll?

- Da die Werte in der rechten unteren Ecke mehr oder weniger alle gleich Null sind, sind auf diese Art längere Runs von Nullen zu erwarten.

Anmerkung: Informationen zu den einzelnen Schritten des JPEG-Algorithmus finden sich in [STE93]

2 Aufgabe - MPEG

1. Der MPEG Standart sieht die drei Bildtypen *I-, P- und B-Frame vor*. Beschreiben sie kurz die drei Frametypen.

   - I-Frames

     Zur Kodierung der I-Frames oder Intra-Coded Frames werden keine Informationen benachbarter Bilder herangezogen. Der Algorithmus benutzt nur bildinterne Informationen. Das Verfahren beruht auf dem JPEG-Algorithmus.

   - P-Frames

     Zur Kodierung von P-Frames (Predicted Frames) werden Informationen vorhergehender Bilder benutzt. I.d.R. sind zwei aufeinanderfolgende Bilder eines Films sehr ähnlich. Ein Bild lässt sich meist weitgehend aus dem vorhergehenden Bild erstellen, indem man die Position von Objekten in dem Bild als Vektor in Relation zum vorhergehenden Bild angibt. Der Standard sieht vor, dass die Einzelbilder eines Films in sog. Makroblöcke zerlegt werden. Die Position jedes einzelnen Makroblocks wird dann im vorhergehenden I-Frame gesucht. Die "bestpassende" Position wird zurückgegeben.

   - B-Frames

     Zur Kodierung der B-Frames (Bidirectional Predicted Frames) werden ebenfalls Bewegungsvektoren benutzt. Hier kann sich der Vektor jedoch sowohl auf das vorhergehende als auch auf das nachfolgende I-Frame beziehen. Es ist auch möglich die Position aus Vorgänger und Nachfolger zu interpolieren.

2. **P-Frames.**

   P-Frames bestehen aus Bewegungsvektoren, die die Position eines Makroblocks (16×16 Pixel) in Relation zum vorherigen I-Frame angeben.

Der Standard enthält jedoch keinerlei Algorithmen zur Berechnung der Bewegungsvektoren.

Der *Exhaustive-Search* Algorithmus ist eine naive Methode: Hier wird zunächst ein Suchraum um die Ausgangsposition festgelegt. Dieser wird durch eine maximale Abweichung p in horizontaler und vertikaler Richtung definiert. Anschließend werden alle Positionen innerhalb des Suchraumes "ausprobiert" und die bestpassende Position zurueckgegeben.

Geben Sie den Exhaustive-Search Algorithmus in Pseudocode an.

```
Frames F[n];

FOR all Frames f DO
  minErr= MAXINT;
  FOR all  Makroblocks M DO
    FOR all Rows row within [M.Row-p; M.Row+p] DO
      FOR all Columns col within [M.Col-p; M.Col+p] DO
        IF (CompareMakroblocks(F[f-1], F[f]) < minErr)
            minErr= CompareMakroblocks(F[f-1], F[f]);
            minRow= Row;
            minCol= Col;

  print M, minCol, minRow, minErr;
```

Bestimmen Sie die Zeitkomplexität des Algorithmus.

Die Zeitkomplexität ergibt sich aus der Anzahl der Frames n, der Anzahl der Makroblocks m, und dem Suchraum p:

$$Komplexitaet = O(n * m * p * 4);$$

3. Bonusaufgabe

Überlegen Sie sich eine "schnellere" Alternative zu Exhaustive-Search.

Im folgenden wird der PHODS-Algorithmus von Chen et al. (1991) beschrieben[CRGP94].

Der Algorithmus betrachtet die horizontale und vertikale Suchrichtung zunächst unabhängig. Von einem Suchzentrum aus berechnet der Algorithmus den Fehler in je einem Makroblock rechts und links sowie oberhalb und unterhalb des Suchzentrums. Des weiteren wird der Makroblock im Suchzentrum untersucht. In horizontaler Richtung wird nun aus den drei Postionen "Rechts", "Links" und "Mitte" der Makroblock mit dem kleinsten Fehler gewählt (analog in vertikaler Richtung). Aus den Positionen der beiden gewählten Makroblöcke wird nun ein Vektor gebildet, der auf das neue Suchzentrum zeigt. Der Algorithmus startet nun von diesem Suchzentrum aus erneut. Dabei werden Makroblöcke mit immer kleiner werdendem Abstand vom Suchzentrum aus zum Vergleich herangezogen. Der Algorithmus terminiert, wenn der Vektor die Länge Null besitzt.

Der Algorithmus ermittelt nur dann immer eine optimale Lösung, wenn der Suchfehler monoton zunimmt, je weiter die Blöcke vom optimalen Punkt entfernt sind. Dies ist natürlich in der Realtität nur selten der Fall, weshalb die Lösung häufig suboptimal ist.

Literatur

[STE93] Ralf Steinmetz, *Multimedia-Technologie - Einführung und Grundlagen.* Springer, Heidelberg, Berlin, 1993. ISBN 3-540-56724-0

[CRGP94] Eric Chan, Arturo Rodriguez, Rakeshkumar Gandhi, and Sethuraman Panchanathan. *Experiments on block-matching techniques for video coding*. Multimedia Systems (ACM-Press & Springer-Verlag), 2(5):228–241, December 1994.