Multimedia Technologie und Sicherheit
WS 2010/2011

Universität Passau
Mario Döller und David Coquil

Projekt 2

Thema: Erkennung von Copy-Move Fälschung in Bildern

1 Ziel

Ziel dieses Projekts ist die praxisorientierte Auseinandersetzung mit dem Vorlesungsstoff durch die Implementierung eines Algorithmus zur Erkennung von *Copy-Move* Fälschungen in Bildern. Der zu implementierende Algorithmus ist der **Robust Match** von Fridrich et al., welcher im Paper *Detection of Copy-Move Forgery in Digital Images* (siehe Literatur und Projektsverzeichnis in Studip) beschrieben wird.
Neben der korrekten Implementierung (siehe Details in der nachfolgenden Sektion), ist ein weiterer Beurteilungspunkt die performante Umsetzung des Algorithmus. Zu diesem Zweck messen Sie die Verarbeitungsgeschwindigkeit Ihres Algorithmus (ohne Bild laden bzw. Ergebnis anzeigen) anhand des gegebenen Beispielsbildes *EncampmentSelfTamp.bmp* und geben diese auf die Konsole aus.
Das Projekt wird in einer Endpräsentation gruppenweise vorgestellt. Diese Präsentation umfasst eine Beschreibung der eigenen Implementierung und der gewählten Optimierungsschritte. Zur Evaluierung der Performance wird jede Abgabe unter gleichen Bedingungen während der Präsentation getestet.

2 Hinweise zur Implementierung

2.1 Eingangs-/Ausgangsparameter

Ihr Tool sollte folgende Eingabeparameter verwenden:

- `image` Pfad zur Bild-Datei
- `quality` Qualitätsparameter der DCD-Quantisierung (siehe unten und p.9 im Referenzpaper), reelle Zahl $\in]0, 1]$
- `threshold` Schwellwert zur Erkennung von doppelten Regionen (siehe p.8 im Referenzpaper)

Als Ausgang sollte das Tool eine grafische Darstellung der identifizierten kopierten Regionen anzeigen.

2.2 Allgemeiner Ablauf

Der Algorithmus wird in folgender Tabelle zusammengefasst.

```
Bilddatei laden
Matrize initialisieren
for each (16x16-Block) do
        DCT-Koeffizienten berechnen
        Koeffizienten in die nächste Zeile der Matrize schreiben
end
Matrize lexikographisch sortieren
for each Paar von aufeinander folgenden identischen Zeilen
        Shift-Vektor-Zählerwerk inkrementieren
end
for each Shift-Vektor mit (Zählerwerk > threshold)
        Alle Ursprungs- und Zielregionen markieren
end
Ergebnis anzeigen
```

2.3 Bild-Datei laden

Hier sollte das Bild in ein Format geladen werden, welches den direkten Zugang auf die Pixelwerte ermöglicht (siehe Java tutorial in Literatur). Gegebenenfalls sollten RGB-Farbwerte in Graustufen anhand der klassischen Formel umgewandelt werden:
$I = 0.299R + 0.587G + 0.114B$

2.4 2-Dimensionelle Diskrete Cosinus Transformation

Für diesen Teil des Programs können Sie gerne eine bereits existierende Implementierung der DCT verwenden (z.B.: JPEG-Kodierung). Vorsicht: Achten Sie auf die korrekte Verwendung der Blockgrößen: der Fridrich-Ansatz verwendet nicht 8x8 sondern 16x16 Pixelblöcke. Sie sollen also die generalisierte Formel der 2-dimensionellen DCT-II benutzen, welche wie folgt lautet:

$$F(u,v) = \alpha(u)\alpha(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) \cos\left(\frac{\pi(2i+1)u}{2N}\right) \cos\left(\frac{\pi(2j+1)v}{2N}\right) \quad (1)$$

mit:
N = Dimension des Blocks; N=8 für den JPEG-Codec, N=16 im Referenz-Paper
$u, v \in [0..N-1]$ = Indexe der Koeffizientenmatrix,
$f(i,j)$ = Wert des Blockpixels an der Stelle (i,j),
$\alpha(i) = \begin{cases} \frac{1}{\sqrt{N}} & \text{für i = 0,} \\ \frac{1}{\sqrt{2N}} & \text{sonst} \end{cases}$

Die Ergebnisse sollen dann quantisiert werden. Dazu wird eine 16x16 Quantisierungsmatriz wie im Paper spezifiziert (p. 9) aus der Standard 8x8 JPEG-Matrize berechnet. Die JPEG-Matrize wird in Abbildung 2.4 dargestellt. Danach sollen die Koeffizienten durch den Eingabeparameter `quality` dividiert werden und gerundet werden.

Die Berechnung der DCT-Koeffizienten ist relativ rechenintensiv, es liegt hier Potenzial, um

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Abbildung 1: Quantisierungsmatrize

die Geschwindigkeit des Programms zu optimieren. Es lässt sich bemerken, dass manche Teile von Gleichung (1) fix sind, und vorab berechnet werden könnten.

2.5 Lexikographisches Sortieren

Fürs lexikographische Sortieren der Matrize wird empfohlen, die *sort* Methode des Java Collections Frameworks anzuwenden.

2.6 Shift-Vektoren

Vergessen Sie nicht, die Shift-Vektoren zu normalisieren! (Siehe p.8 im Referenz-Paper).

3 Test-Datensatz

Die Musterbilder des Referenz-Papers sind im Projektverzeichnis verfügbar. Das Bild ”output.png” zeigt das Ergebnis der Analyse von ”EncampmentSelfTamp.bmp” mit Parametern: quality = 0.5, threshold = 10.

Literatur

[1] Jessica Fridrich, David Soukal, and Jan Lukas, *Detection of Copy-Move Forgery in Digital Images*, Digital Forensics Research Workshop, 2003.

[2] *Java Tutorial: Working with images* `http://download.oracle.com/javase/tutorial/2d/images/index.html`

[3] *Java Tutorial: Collections framework* `http://download.oracle.com/javase/tutorial/collections/interfaces/index.html`

[4] Syed Ali Khayam, *The Discrete Cosine Transform (DCT): Theory and Application*, Department of Electrical & Computer Engineering, Michigan State University, `http://www.egr.msu.edu/waves/people/Ali_files/DCT_TR802.pdf`