

Untersuchung verschiedener Einflüsse auf die Performance eines MPEG Encoders

Projekt für die Vorlesung
“Computer Systems Performance Analysis and Benchmarking”
Wintersemester 02/03, ETH Zürich

Ingo Opper
oingo@student.ethz.ch
Rechnergestützte Wissenschaften

Inhalt

Inhalt	1
1. Einleitung	2
2. ezMPEG	3
2.1 MPEG-1 Grundlagen	3
2.2 Metriken	4
2.3 Workload	4
3. System	5
4. Parameter	6
4.1 Compiler	6
4.2 DCT Algorithmus	6
4.3 Festplattenzugriff	6
5. Ablauf	7
6. Analyse und Folgerungen	8
6.1 Analyse	8
6.2 Folgerungen	9
7. Referenzen	11
Anhang A: DCT Algorithmen	12
Anhang B: Resultate	14

1. Einleitung

Das Ziel dieses Projektes ist es herauszufinden, was für einen Einfluss der Compiler, verwendete Algorithmen und Festplattenzugriffe auf die Geschwindigkeit eines Encodingprozesses von einem MPEG-1 Video Stream haben.

Als MPEG-Encoder wird ezMPEG [1] verwendet. Dabei handelt es sich um eine Open-Source Implementation des ISO/IEC 11172 Standards [2]. Dieser Standard besteht im wesentlichen aus 4 Teilen (Systems, Video, Audio, Compliance testing) wobei nur der zweite Teil zum Tragen kommt und davon nur die Codierung von I-Frames (siehe Abschnitt 2.1).

2. ezMPEG

Bei ezMPEG handelt es sich um eine Open-Source MPEG-1-Video Encoder API, die komplett in C geschrieben ist. Dadurch kann sie auf jedem System eingesetzt werden, das über einen Standard C Compiler verfügt. Bei der Implementation der API wurde nicht auf schon vorhandene Bibliotheken oder Code Segmente von anderen MPEG-Encodern zurückgegriffen, sondern alles wurde von Grund auf neu geschrieben.

In diesem Projekt wird die Version 0.1a von ezMPEG benutzt. Diese Version implementiert zurzeit nur die Codierung von I-Frames und erzeugt einen reinen MPEG-1 Video-Stream. Audio-daten werden nicht berücksichtigt.

2.1 MPEG-1 Grundlagen

MPEG-1 wurde von der Motion Pictures Expert Group [3] entwickelt, um Ton und bewegte Bilder möglichst platzsparend zu speichern und zu transportieren, ohne dass die Qualität zu sehr leidet, indem redundante Informationen im Datenstrom reduziert werden. Im Folgenden beschränkt sich die Beschreibung der Kompression nur auf die Bilddaten.

Die Menge der zu speichernden/transportierenden Daten kann auch noch durch Weglassen von Informationen, die schon in einem vorherigen oder kommenden Bild vorkommen, reduziert werden. Dazu wurden in MPEG drei (vier) Bildtypen definiert: I-Frames (alle nötigen Informationen sind vorhanden), P-Frames (ein Teil der Informationen bezieht sich auf ein voriges I- oder P-Frame), B-Frames (ein Teil der Informationen bezieht sich auf ein voriges und kommendes I- oder P-Frame), D-Frames (keine Information). So besteht ein MPEG Video aus Sequenzen von I-, P- und B-Frames. Beispiele von gültigen Sequenzen: IIII, IPPPP, IBBPBBPBB.

Bei der Kompression werden verschiedene Eigenschaften der menschlichen Wahrnehmung ausgenutzt. So reagiert das menschliche Auge stärker auf Helligkeitsänderungen als auf Farbton-änderungen. Diese Tatsache wird in einem ersten Schritt der Kompression ausgenutzt. Dort wird das Bild in einen Farbraum umgewandelt, der die Helligkeits- und Farbinformationen trennt. Der zweite Schritt besteht darin, die hohen Frequenzen aus den Bildinformation zu filtern, da diese vom Auge nicht so stark wahrgenommen werden wie die niedrigen Frequenzen. Im dritten und letzten Schritt werden häufig vorkommende Zahlenkombinationen als kurze Codes, und seltene Kombinationen als längere Codes gespeichert, was wiederum eine Kompression bedeutet.

Da es zu aufwändig ist, diese drei Schritte auf dem ganzen Bild auszuführen, wird das Bild in Blöcke unterteilt. Jeder Block ist 8x8 Pixel gross. Für die weitere Verarbeitung werden die Blöcke zu einem 16x16 Pixel grossen Macroblock zusammengefasst. So wären es 12 Blöcke pro Macroblock (pro Farbe 4 Blöcke). Durch die Umwandlung des Bildes vom RGB in den YCrCb Farbraum werden Helligkeitsinformation von den Farbinformationen getrennt. Jetzt greift der erste Schritt der Kompression, denn in einem Macroblock werden nur 6 Blöcke gespeichert, wobei 4 Blöcke den Y-Farbkanal (Helligkeit) und die restlichen zwei Blöcke den Cr- bzw. den Cb-Farbkanal (Farbton) über den ganzen Macroblock gemittelt beinhalten. D.h. in einem Macroblock sind 64 Helligkeitswerte und 32 Farbwerte gespeichert. Dadurch wurden die vorhandenen Information schon um 50% reduziert, allerdings mit Verlusten in der Bildqualität, die jedoch kaum wahrnehmbar sind.

Im folgenden Schritt wird auf jeden Block im Macroblock die Diskrete Cosinus Transformation (DCT, symmetrischer Teil der Diskreten Fourier Transformation) angewendet. Damit werden die Helligkeits- und Farbinformationen in den Frequenzraum übergeführt. Das führt dazu, dass in jedem 8x8 Block die niedrigen (relevanten) Frequenzen links oben und die hohen Frequenzen rechts unten in der Matrix stehen. Das macht es sehr einfach die hohen Frequenzen herauszufiltern. Dies geschieht mit einer Gewichtungsmatrix, die jede Zahl im transformierten Block ganzzahlig teilt (Quantisierung). Dadurch werden einige Frequenzen zu 0. Die Gewichtungsmatrix ist so aufgebaut, dass links oben kleine und rechts unten grosse Zahlen stehen (jene Zahlen wurden empirisch aus psycho-visuellen Tests ermittelt). Wird vor der Division die Gewichtungsmatrix mit einem Qualitätsfaktor multipliziert, werden entsprechend mehr hohe Frequenzen auf 0 gesetzt. D.h. ein hoher Qualitätsfaktor ergibt eine niedrige Qualität.

Durch die Gewichtung der Frequenzen sind viele Einträge des Blocks zu 0 gesetzt worden, deshalb macht es keinen Sinn, jede 0 zu speichern. Dazu wird die Matrix in eine serielle Form (Zig-Zag) gebracht und die vorhandenen Werte werden zuerst RLE- und die dabei entstehenden Run/Value-Paare Huffman-kodiert. Die entsprechenden Huffman-Codes müssen nicht bei jedem Block neu erzeugt werden, sondern sind im Standard vorgegeben und ermöglichen dadurch eine schnelle Verarbeitung der Daten.

2.2 Metriken

Die relevante Metrik bei der Kompression zu einem MPEG Datenstrom ist, wieviele Bilder pro Sekunde verarbeitet werden können. Dies kann in einem Bereich von weniger als einem Bild über 25 Bilder (Echtzeit) oder mehr liegen. Es gilt herauszufinden, was diese Grösse auf einem gegebenen System am meisten beeinflusst.

2.3 Workload

Der Workload besteht aus einer Sequenz von 125 Einzelbildern im PPM-Format mit der Dimension von 320x240 Pixel, wobei pro Pixel 3 Byte für die RGB-Farbinformationen verwendet werden. Der MPEG-Encoder generiert daraus eine Videosequenz von 5 Sekunden Länge. Der Qualitätsfaktor ist fest auf 8 eingestellt.

3. System

Im Folgenden werden die relevanten Parameter aufgelistet. Diese ändern sich während des gesamten Benchmarks nicht.

Hardware:

- IBM ThinkPad 600
- CPU: Intel Mobile Pentium II 300MHz
- RAM: 192MB
- Harddisk: IBM Travelstar 30GN-20 20GB

Software:

- Microsoft Windows 2000 Professionel mit Service Pack 2
- ezMPEG 0.1a

Während des Benchmarks laufen nur die nötigsten Prozesse im Hintergrund, d.h. Virens Scanner o.ä. sind deaktiviert. Netzwerk- und Grafikkarte haben keinen Einfluss auf die Geschwindigkeit. Der Benchmark läuft als Konsolenanwendung in einem DOS-Fenster.

4. Parameter

Es werden folgende 3 Parameter während des Benchmarks geändert

- Compiler
- DCT Algorithmus
- Festplattenzugriffe ein/aus

4.1 Compiler

Diese 3 Compiler werden verwendet:

- DJGPP gcc Version 3.2.1 [4]
Freeware und Open-Source C/C++ Compiler für 32-bit DOS
Compiler-Flags: -O3
- LCC-Win32 Version 3.8 [5]
Freeware C-Compiler für Windows
Compiler-Flags: -O
- Microsoft C/C++ Compiler Version 12.0 [6]
Kommerzieller C/C++ Compiler für Windows von Microsoft
Teil des Microsoft Visual Studio 6
Compiler-Flags: Release-Settings

4.2 DCT Algorithmus

Für die Transformation der Farbwerte in den Frequenzraum werden diese beiden DCT Algorithmen verwendet:

- Implementation der direkten 2D DCT
- Implementation einer doppelt angewandten optimierten 1D DCT

Beide DCT Algorithmen rechnen mit einfacher Fließkommagenauigkeit. Anhang A gibt eine Beschreibung der Algorithmen.

4.3 Festplattenzugriff

Dieser Parameter entscheidet, ob während des Benchmarks auf die Platte zugegriffen werden soll oder nicht. Damit soll der Einfluss des Einlesens der benötigten Bilddaten und das Schreiben des MPEG Datenstroms bestimmt werden. Festplattenzugriffe werden dadurch vermieden, dass alle Bilddaten vorher in den Speicher gelesen werden und der Codierungsprozess erst danach beginnt. Die Zeitmessung beginnt mit dem Codierungsprozess.

5. Ablauf

Alle Kombinationen der Faktoren werden getestet.

- | | | | |
|-----|--------------------|----------|--------------------------|
| 1. | DJGPP Compiler | - 2D DCT | - Festplattenzugriff ein |
| 2. | DJGPP Compiler | - 2D DCT | - Festplattenzugriff aus |
| 3. | DJGPP Compiler | - 1D DCT | - Festplattenzugriff ein |
| 4. | DJGPP Compiler | - 1D DCT | - Festplattenzugriff aus |
| 5. | LCC Compiler | - 2D DCT | - Festplattenzugriff ein |
| 6. | LCC Compiler | - 2D DCT | - Festplattenzugriff aus |
| 7. | LCC Compiler | - 1D DCT | - Festplattenzugriff ein |
| 8. | LCC Compiler | - 1D DCT | - Festplattenzugriff aus |
| 9. | Microsoft Compiler | - 2D DCT | - Festplattenzugriff ein |
| 10. | Microsoft Compiler | - 2D DCT | - Festplattenzugriff aus |
| 11. | Microsoft Compiler | - 1D DCT | - Festplattenzugriff ein |
| 12. | Microsoft Compiler | - 1D DCT | - Festplattenzugriff aus |

Nach jedem Durchlauf wird die benötigte Laufzeit in Sekunden ausgegeben. Daraus kann berechnet werden, wie lange es dauerte, ein Frame zu kodieren. Jede Kombination wird dreimal durchlaufen und die entsprechenden Resultate werden gemittelt.

6. Analyse und Folgerungen

Die Resultate des Benchmarks sind in Anhang B aufgelistet. Der Benchmark bestand aus dem Durchtesten aller 12 Kombinationen von Compiler, DCT und Festplattenzugriffen. Jeder einzelne Test wurde 3mal ausgeführt. Die 3 Messergebnisse sind bei der Auflistung in Anhang B in den entsprechenden Abschnitten zusammengefasst.

Bei einem ersten Vergleich der Zahlen der unterschiedlichen Compiler, fällt der LCC-Win32 ziemlich aus der Reihe. Dessen erbrachten Frameraten (Test 5 - 8) sind bis zu 1.5mal kleiner als die der anderen beiden Kandidaten. Daraus lässt sich schon schliessen, dass dort die Wahl des Compilers den grössten Teil des Effektes erklärt. Der Microsoft C/C++ Compiler ist der schnellste im Testfeld, aber dicht gefolgt vom DJGPP gcc. Bei diesen beiden sind die Unterschiede nicht sehr gravierend und die Abhängigkeiten von den Faktoren auf die Performance muss noch genauer untersucht werden.

6.1 Analyse

Im Folgenden wird die Abhängigkeit von den Faktoren auf die erzeugten Frameraten untersucht. Dabei werden die Zahlen vom Microsoft Compiler mit den Zahlen vom DJGPP gcc verglichen. Der LCC-Win32 bleibt aussen vor, da sich dessen Zahlen durch die Wahl des Compilers erklären lassen. Die Verhältnisse der Werte der Tests untereinander sind bei allen verwendeten Compilern etwa gleich. Deswegen erklärt die auf zwei Compiler eingeschränkte Analyse auch die Effekte bei den Tests mit dem LCC-Win32.

Betrachtet man die erreichten Frameraten des ersten Durchlaufs aller Tests, so stellt man fest, dass diese in den meisten Fällen immer etwas niedriger ausgefallen sind, als die der restlichen Durchgänge. Dort zeigen die Werte nur geringe Abweichungen zueinander. Dieser Sprung in den Frameraten kann durch Caching-Effekte erklärt werden. Beim ersten Durchlauf wird der Code des Programmes und ein Teil der zu bearbeitenden Daten in den Speicher geladen, was bei den folgenden Durchläufen nicht mehr nötig ist. Dieses Verhalten ist dem Betriebssystem anzurechnen und wird in der Analyse berücksichtigt.

Um den Caching-Effekten des Betriebssystems Rechnung zu tragen, werden die Zahlen in Form eines $2^3 \times 3$ und zwei 2^3 Factorial Designs analysiert. Im ersten Fall werden die erreichten Frameraten eines Tests gemittelt und die Effekte der einzelnen Faktoren berechnet. In den beiden anderen Designs werden jeweils die Werte des ersten und des letzten Durchlaufs getrennt voneinander betrachtet, um zu sehen, welchen Einfluss der Cache des Betriebssystems auf die Faktoren nimmt. Dazu werden folgende Variablen eingeführt:

$$\text{Faktor A: } x_A = \begin{cases} 1 & \text{DJGPP gcc} \\ -1 & \text{Microsoft C/C++ Compiler} \end{cases}$$

$$\text{Faktor B: } x_B = \begin{cases} 1 & \text{2D DCT} \\ -1 & \text{1D DCT} \end{cases}$$

$$\text{Faktor C: } x_C = \begin{cases} 1 & \text{Festplattenzugriff ein} \\ -1 & \text{Festplattenzugriff aus} \end{cases}$$

Diese Wahl der Variablen führt zu folgender Tabelle:

Tabelle 6.1

I	A	B	C	AB	AC	BC	ABC	y	Mean \bar{y}	y_1	y_3
1	1	1	1	1	1	1	1	(2.18, 2.76, 2.76)	2.57	2.18	2.76
1	1	1	-1	1	-1	-1	-1	(2.98, 2.98, 2.98)	2.98	2.98	2.98
1	1	-1	1	-1	1	-1	-1	(2.56, 3.81, 3.81)	3.40	2.56	3.81
1	1	-1	-1	-1	-1	1	1	(3.65, 4.24, 4.24)	4.04	3.65	4.24
1	-1	1	1	-1	-1	1	-1	(2.78, 3.39, 3.39)	3.19	2.78	3.39
1	-1	1	-1	-1	1	-1	1	(3.55, 3.56, 3.56)	3.56	3.55	3.56
1	-1	-1	1	1	-1	-1	1	(3.18, 4.08, 4.08)	3.78	3.18	4.08
1	-1	-1	-1	1	1	1	-1	(4.34, 4.32, 4.35)	4.34	4.34	4.35
27.86	-1.88	-3.26	-1.98	-0.52	-0.12	0.42	0.04		Total		
3.48	-0.24	-0.41	-0.25	-0.07	-0.02	0.05	0.00		Total/8		
25.22	-2.48	-2.24	-3.82	0.14	0.04	0.68	-0.01		Total		
3.15	-0.31	-0.28	-0.48	0.02	0.01	0.09	-0.00		Total/8		
29.16	-1.58	-3.78	-1.08	-0.84	-0.22	0.30	0.12		Total		
3.65	-0.20	-0.47	-0.14	-0.11	-0.03	0.04	0.02		Total/8		

Der Faktor B (die Wahl des DCT-Algorithmus) hat mit 43.19% den grössten Einfluss auf die Performance, wenn man die gemittelten Werte betrachtet. Die Wahl des Compilers und die Festplattenzugriffe spielen mit 14.79% und 16.08% eher eine untergeordnete Rolle. Die Interaktionen zwischen den Faktoren kommen mit weniger als je 0.12% nicht zum Tragen. Die restlichen 23.90% können nicht zugeordnet werden und werden dem Fehler angerechnet.

Das Bild sieht etwas anders aus, wenn nur die Zahlen des ersten Durchlaufs (y_1) in Betracht gezogen werden. Dort hat, wie es zu erwarten war, die Festplattenaktivität mit 55.76% den grössten Einfluss. Die Relevanz der Faktoren A und B reduziert sich auf 23.33% und 19.09%. Alle Interaktionen, bis auf DCT/Festplatte (1.82%), haben mit 0.00% gar keinen Einfluss.

Die Werte unter Cache-Einfluss (y_3) zeigen ein extremeres Bild. Dort schlägt die Wahl des DCT-Algorithmus mit 74.68% zu Buche. Der Compiler hat immerhin noch einen Einfluss von 13.50% und die Festplatte nur 6.75%. Die Interaktion zwischen Compiler und DCT hat mit 4.22% einen recht grossen Anteil. Die restlichen Interaktionen liegen unter 0.50%.

6.2 Folgerungen

Im Gesamten ist die Wahl des DCT-Algorithmus für die Performance des Encoding-Prozesses entscheidend. Dort gibt es viele Möglichkeiten noch mehr Leistung rauszuholen. Zum Beispiel lassen sich verschiedene Fast-DCT's implementieren, oder man beschränkt sich auf ganze Zahlen während der Berechnung. Allerdings sind diese Verfahren nur Näherungen, jedoch lassen die Spezifikationen einen kleinen Spielraum in der Genauigkeit zu.

Der Compiler hat einen geringen Einfluss auf die Leistung. Nur der LCC-Win32 fällt da ein wenig aus der Rolle. Durch Compiler-spezifische Optimierungen und Flags liessen sich da sicher noch einige Frames pro Sekunde rauskitzeln. Der Freeware und Open-Source Compiler DJGPP gcc kann es gut mit dem kommerziellen Produkt aus dem Hause Microsoft aufnehmen.

Die Festplatte ist ein Flaschenhals für die Performance. Darum sollten Zugriffe, wenn möglich, vermieden werden. In Kombination mit Caches wird der Einfluss geringer, ist aber noch spürbar. Kombinationen der Faktoren haben geringen bis keinen Einfluss auf die Leistung des Encoders.

Die höchsten Frameraten erreicht der Encoder zusammen mit der optimierten 1D DCT und mit dem C/C++ Compiler von Microsoft. Falls man nicht auf kommerzielle Produkte setzen will, ist der weit verbreitete gcc zu empfehlen. Wenn noch ein schnellerer Prozessor hinzukommt, wird auch ein Echtzeit-Encoding (oder besser) möglich sein.

7. Referenzen

[1] ezMPEG MPEG Encoder API Homepage
<http://www.sourceforge.net/projects/ezmpeg/>

[2] ISO Homepage
<http://www.iso.ch/>

[3] MPEG Homepage
<http://mpeg.telecomitalialab.com/>

[4] DJGPP Homepage
<http://www.delorie.com/djgpp/>

[5] LCC-Win32 Homepage
<http://www.cs.virginia.edu/~lcc-win32/>

[6] Microsoft Visual Studio Homepage
<http://msdn.microsoft.com/vstudio/>

Anhang A: DCT Algorithmen

Die 2-dimensionale 8x8 DCT ist definiert als

$$F(u, v) = c(u) c(v) \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos \frac{\pi(2x+1)u}{16} \cos \frac{\pi(2y+1)v}{16}, \quad u, v = 0 \dots 7$$

mit $c(k) = \frac{1}{2\sqrt{2}}$ für $k = 0$ und $c(k) = \frac{1}{2}$ sonst. Die Variablen x und y entsprechen den Pixelkoordinaten des 8x8 Blocks, wobei $f(x, y)$ den Farbwert an diesen Koordinaten bezeichnet. $F(u, v)$ ist dann der Wert im Frequenzraum an den Koordinaten u und v .

In der Implementation dieses Algorithmus werden die Cosinus-Werte nicht bei jeder Transformation neu berechnet, sondern werden zu Beginn einmal in einem Array abgespeichert. Dies führt schon zu einem erheblichen Geschwindigkeitszuwachs, da nur noch auf die Einträge des Arrays zugegriffen werden muss. Allerdings sind dann pro Frequenz immer noch 194 Multiplikationen durchzuführen, was bei einem ganzen Block auf 12288 Multiplikationen kommt.

Ein Bild der Grösse 320x240 Pixel besteht aus 300 Macroblöcken und in jedem Macroblock sind 6 Blöcke. D.h. pro Bild muss die DCT 1800-mal ausgeführt werden. Das ist ein enormer Rechenaufwand, der unbedingt reduziert werden muss, um eine zumutbare Geschwindigkeit beim Encoding zu erreichen.

Um den Rechenaufwand weiter zu minimieren, kann die obige Formel trigonometrisch vereinfacht werden. Dazu betrachtet man die 1-dimensionale DCT. Das ist erlaubt, weil die DCT separierbar ist, d.h. um eine 2-dimensionale DCT auszuführen, kann auch eine 1-dimensionale DCT zuerst in horizontaler Richtung und dann in vertikaler Richtung berechnet werden. Die folgende Vereinfachung der DCT kommt auch in ezMPEG zur Anwendung.

Die 1-dimensionale DCT ist definiert als

$$X_i = c(i) \sum_{k=0}^7 x_k \cos \frac{\pi(2k+1)i}{16}, \quad i = 0 \dots 7$$

wobei in folgender Rechnung der Faktor $c(i)$ der Einfachheit halber gleich 1 gesetzt wird. Die DCT kann nun in einen geraden und einen ungeraden Anteil zerlegt werden. Zuerst wird der gerade Anteil ($i = 2j$) vereinfacht:

$$X_{2j} = \sum_{k=0}^7 x_k \cos \frac{\pi(2k+1)j}{8}$$

wird in zwei Summen aufgeteilt

$$X_{2j} = \sum_{k=0}^3 x_k \cos \frac{\pi(2k+1)j}{8} + \sum_{k=0}^3 x_{k+4} \cos \frac{-\pi(2(k+4)+1)j}{8}$$

Das Minuszeichen im Cosinus der zweiten Summe hat keine Auswirkung, da die Cosinus-Funktion symmetrisch ist, hilft aber bei der Umformung. Im folgenden Schritt wird die Summationsreihenfolge des zweiten Summanden vertauscht und das Argument im Cosinus wird etwas umgeformt

$$X_{2j} = \sum_{k=0}^3 x_k \cos \frac{\pi(2k+1)j}{8} + \sum_{k=0}^3 x_{7-k} \cos \left(\frac{-\pi(2(-1-k)+1)j}{8} - 2\pi j \right)$$

was sich weiter vereinfacht zu (unter Verwendung von $\cos(a - b) = \cos(a) \cos(b) + \sin(a) \sin(b)$)

$$X_{2j} = \sum_{k=0}^3 x_k \cos \frac{\pi(2k+1)j}{8} + \sum_{k=0}^3 x_{7-k} \cos \frac{\pi(2k+1)j}{8}$$

Daraus folgt ein Ausdruck einer DCT der Länge 4:

$$X_{2j} = \sum_{k=0}^3 (x_k + x_{7-k}) \cos \frac{\pi(2k+1)j}{8}, \quad j = 0 \dots 3$$

Die selbe Umformung kann auch für den ungeraden Teil ($i = 2j+1$) der DCT gemacht werden, allerdings ist diese etwas komplizierter:

$$X_{2j+1} = \sum_{k=0}^7 x_k \cos \frac{\pi(2j+1)(2k+1)}{16}$$

wird wieder in zwei Teile geteilt und die Argumente werden ausmultipliziert

$$X_{2j+1} = \sum_{k=0}^3 x_k \cos\left(\frac{\pi(2k+1)j}{8} + \frac{\pi(2k+1)}{16}\right) + \sum_{k=0}^3 x_{k+4} \cos\left(\frac{\pi(2(k+4)+1)j}{8} + \frac{\pi(2(k+4)+1)}{16}\right)$$

Die zweite Summe kann, wie im geraden Anteil, vereinfacht werden

$$\sum_{k=0}^3 x_{k+4} \cos\left(\frac{\pi(2(k+4)+1)j}{8} + \frac{\pi(2(k+4)+1)}{16}\right) = \sum_{k=0}^3 x_{7-k} (-1) \cos\left(\frac{\pi(2k+1)j}{8} + \frac{\pi(2k+1)}{16}\right)$$

Dies führt wiederum zu einer DCT der Länge 4, aber diesmal für den ungeraden Teil:

$$X_{2j+1} = \sum_{k=0}^3 (x_k - x_{7-k}) \cos\left(\frac{\pi(2k+1)j}{8} + \frac{\pi(2k+1)}{16}\right), \quad j = 0 \dots 3$$

Durch diese Trennung der 1-dimensionalen DCT in einen geraden und einen ungeraden Anteil und deren Vereinfachung kann der Rechenaufwand bei der Transformation weiter reduziert werden. Desweiteren müssen nur noch halb so viele Cosinus-Werte berechnet, bzw. in einem Array gespeichert werden.

Anhang B: Resultate

Test 1

Compiler: DJGPP gcc
Flags: -O3
DCT: 2DDCT
Festplatte: ein

Frames: 125
Dauer (clocks): 5215; 4120; 4110
Dauer (secs): 57.307693; 45.274727; 45.164837 (91 clocks pro Sekunde)
Frames pro Sekunde: 2.181208; 2.760922; 2.767640
Sekunden pro Frame: 0.458462; 0.362198; 0.361319

Test 2

Compiler: DJGPP gcc
Flags: -O3
DCT: 2DDCT
Festplatte: aus

Frames: 125
Dauer (clocks): 3805; 3805; 3805
Dauer (secs): 41.813187; 41.813187; 41.813187 (91 clocks pro Sekunde)
Frames pro Sekunde: 2.989488; 2.989488; 2.989488
Sekunden pro Frame: 0.334505; 0.334505; 0.334505

Test 3

Compiler: DJGPP gcc
Flags: -O3
DCT: 1DDCT
Festplatte: ein

Frames: 125
Dauer (clocks): 4440; 2985; 2985
Dauer (secs): 48.791210; 32.802197; 32.802197 (91 clocks pro Sekunde)
Frames pro Sekunde: 2.561937; 3.810720; 3.810720
Sekunden pro Frame: 0.390330; 0.262418; 0.262418

Test 4

Compiler: DJGPP gcc
Flags: -O3
DCT: 1DDCT
Festplatte: aus

Frames: 125
Dauer (clocks): 3110; 2680; 2680
Dauer (secs): 34.175823; 29.450550; 29.450550 (91 clocks pro Sekunde)
Frames pro Sekunde: 3.657556; 4.244403; 4.244403
Sekunden pro Frame: 0.273407; 0.235604; 0.235604

Test 5

Compiler: LCC-Win32
Flags: -O
DCT: 2DDCT
Festplatte: ein

Frames: 125
Dauer (clocks): 97521; 86173; 86214
Dauer (secs): 97.521004; 86.172997; 86.213997 (1000 clocks pro Sekunde)
Frames pro Sekunde: 1.281775; 1.450570; 1.449881
Sekunden pro Frame: 0.780168; 0.689384; 0.689712

Test 6

Compiler: LCC-Win32
Flags: -O
DCT: 2DDCT
Festplatte: aus

Frames: 125
Dauer (clocks): 77802; 77782; 77782
Dauer (secs): 77.802002; 77.781998; 77.781998 (1000 clocks pro Sekunde)
Frames pro Sekunde: 1.606642; 1.607056; 1.607056
Sekunden pro Frame: 0.622416; 0.622256; 0.622256

Test 7

Compiler: LCC-Win32
Flags: -O
DCT: 1DDCT
Festplatte: ein

Frames: 125
Dauer (clocks): 67016; 52355; 52386
Dauer (secs): 67.015999; 52.355000; 52.386002 (1000 clocks pro Sekunde)
Frames pro Sekunde: 1.865226; 2.387547; 2.386134
Sekunden pro Frame: 0.536128; 0.418840; 0.419088

Test 8

Compiler: LCC-Win32
Flags: -O
DCT: 1DDCT
Festplatte: aus

Frames: 125
Dauer (clocks): 43633; 43623; 43623
Dauer (secs): 43.632999; 43.623001; 43.623001 (1000 clocks pro Sekunde)
Frames pro Sekunde: 2.864804; 2.865461; 2.865461
Sekunden pro Frame: 0.349064; 0.348984; 0.348984

Test 9

Compiler: Microsoft C/C++ Compiler
Flags: Release-Settings
DCT: 2DDCT
Festplatte: ein

Frames: 125
Dauer (clocks): 45024; 36923; 36913
Dauer (secs): 45.024002; 36.923000; 36.913002 (1000 clocks pro Sekunde)
Frames pro Sekunde: 2.776297; 3.385424; 3.386341
Sekunden pro Frame: 0.360192; 0.295384; 0.295304

Test 10

Compiler: Microsoft C/C++ Compiler
Flags: Release-Settings
DCT: 2DDCT
Festplatte: aus

Frames: 125
Dauer (clocks): 35120; 35090; 35100
Dauer (secs): 35.120003; 35.090000; 35.100002 (1000 clocks pro Sekunde)
Frames pro Sekunde: 3.559225; 3.562268; 3.561253
Sekunden pro Frame: 0.280960; 0.280720; 0.280800

Test 11

Compiler: Microsoft C/C++ Compiler
Flags: Release-Settings
DCT: 1DDCT
Festplatte: ein

Frames: 125
Dauer (clocks): 39266; 30614; 30624
Dauer (secs): 39.266003; 30.614002; 30.624001 (1000 clocks pro Sekunde)
Frames pro Sekunde: 3.183415; 4.083099; 4.081766
Sekunden pro Frame: 0.314128; 0.244912; 0.244992

Test 12

Compiler: Microsoft C/C++ Compiler
Flags: Release-Settings
DCT: 1DDCT
Festplatte: aus

Frames: 125
Dauer (clocks): 28791; 28911; 28721
Dauer (secs): 28.791002; 28.911001; 28.721001 (1000 clocks pro Sekunde)
Frames pro Sekunde: 4.341634; 4.323614; 4.352216
Sekunden pro Frame: 0.230328; 0.231288; 0.229768