Dokumentation Semesterarbeit

JPEG VIDEO STREAMER

Autoren:	Benjamin Ott, Andy Senn
Betreuender Dozent:	Prof. Dr. Guido M. Schuster
Fach:	Digitale Medien

1 Projekt: Kurzbeschreibung 3

2 JPEG Komprimierungsverfahren 3

2.1 Historie 3

2.2 Kompression mit JPEG 3

2.2.1 Colortransformation & Colorrücktransformation 4
2.2.2 Down- & Up-Sampling 5
2.2.3 ForwardDCT & InverseDCT 6
2.2.4 Quantisierung & Dequantisierung 7
2.2.5 Kodierung & Dekodierung 7

3 Beispiel 8

4 Realisierung 9

4.1 Wie arbeitet das vorliegende Client-Server Programm? 9

4.2 Zugriff und Behandlung der einzelnen Elemente 10

4.2.1 Statisch erzeugte mehrdimensionale Matrizen 10
4.2.2 Dynamisch erzeugte mehrdimensionale Matrizen 11
4.2.3 Matrix-Klasse 11
4.2.4 Buffer und Zeiger 11
4.2.5 Definition der Wertebereiche 11

4.3 Colortransformation 12

4.4 Downsampling 12

4.5 Werte-Aufbereitung für DCT 13

4.6 Diskrete Cosinus Transformation 13

4.7 Quantisierung 14

4.8 Kodierung 14

4.9 Dekodierung 15

4.10 Anhang: Kodierung & Dekodierung 15

5 Resultat 26

5.1 Messresultate 26

5.2 Programm-Beschreibung 27

5.2.1 Mögliche Daten-Quellen 27
5.2.2 Client 27
5.2.3 Server 28

1 Projekt: Kurzbeschreibung

Ziel dieser Semesterarbeit ist es, einen bereits funktionierenden Video-Streamer mit JPEG zu modifizieren. Dies bedeutet, dass wir eine Quellenencodierung respektive Quellendekodierung mit Hilfe der in JPEG benützten DCT-Trasformation implementieren. Bildern die von einer Kamera geschossen werden können somit schneller über ein Netzwerk übertragen werden, da die Datenmenge durch JPEG reduziert wir.

In einer ersten Version werden die Bilder einzeln mit JPEG komprimiert, gesendet und dekomprimiert.
In der zweiten Version soll dann die Kompression erhöht werden, in dem man nur noch die Daten der Bildsegmente sendet, bei denen sich was geändert hat. Um dies zu realisieren, müssen die einzelnen Bilder in eben solche Segmente aufgeteilt werden und mit dem vorhergehenden Bildsegment verglichen werden. Bei einer Abweichung grösser als eine Toleranz, wird das Segment komprimiert und gesendet, ansonsten wird nur die Information gesendet, dass das vorherige Segment nochmals benützt werden kann. Somit wird das Netzwerk enorm entlastet, wenn sich vor der Kamera nichts oder nur wenig ändert.

2 JPEG Komprimierungsverfahren

2.1 Historie

Anfang der achtziger Jahre gab es erste Aktivitäten zur Definition eines Standards für die Codierung von Farbbildern. Innerhalb der ISO (International Organization for Standardization) wurde eine **P**hotographics **E**xperts **G**roup mit dem Ziel gebildet, die Entwicklung eines progressiven Kompressionsverfahrens voranzutreiben. Ähnliche Bestrebungen gab es auch in der CCITT (International Telegraph and Telephone Consultative Committee, heute: ITU-T: International Telecommunications Union-Telecommunication Sector). 1986 schlossen sich die Arbeitsgruppen zur Joint Photographic Experts Group (JPEG) zusammen. Seither wurden verschiedene Komprimierungsverfahren entwickelt.

2.2 Kompression mit JPEG

Um die Datengröße zu verringern, werden Grafiken komprimiert. Komprimierung kann verlustlos oder verlustbehaftet sein. Früher wollte man die Datenmenge durch zusammenfassen gleicher Daten möglichst klein halten. Nach all diesen Verfahren konnte man das Bild zu 100% rekonstruieren. Die Qualität einer Komprimierung zeigte sich durch die Datenmenge die gespart werden konnte. Heute geht man einen Schritt weiter und entwickelt auch Verfahren die durchaus Informationen verlieren. Das hat jedoch den Vorteil, dass man jedes Bild beliebig klein machen kann. Natürlich muss man den Qualitätsverlust in Kauf nehmen. Bei der JPEG Komprimierung ist beides möglich. Die Komprimierungsart die wir ausgewählt haben, ist verlustbehaftet und basiert auf der Baseline DCT (Diskrete Kosinus-Transformation). Sie ist in der Lage Bilddaten mit einer Genauigkeit von 8 oder 12 Bits pro Bildpunkt zu verarbeiten. Insgesamt können 255 Komponenten, wie zum Beispiel Farbkomponenten, codiert werden.

Die Komprimierung läuft in folgenden 5 Grundschritten ab:

1. Colortransformation (RGB → YCrCb)
2. Down-Sampling (4:4:4 → 4:1:1)
3. Diskrete Cosinus Transformation (DCT)
4. Quantisieren der DCT-Koeffizienten
5. Kodieren der Koeffizienten (Huffman-Code)

Die Dekomprimierung läuft in folgenden 5 Grundschritten ab:

1. Decodieren der Koeffizienten (Huffman-Code)
2. Dequantisieren der iDCT-Koeffizienten
3. inverse Diskrete Cosinus Transformation (iDCT)
4. Up-Sampling (4:1:1 → 4:4:4)
5. Colorrücktransformation (YCrCb → RGB)

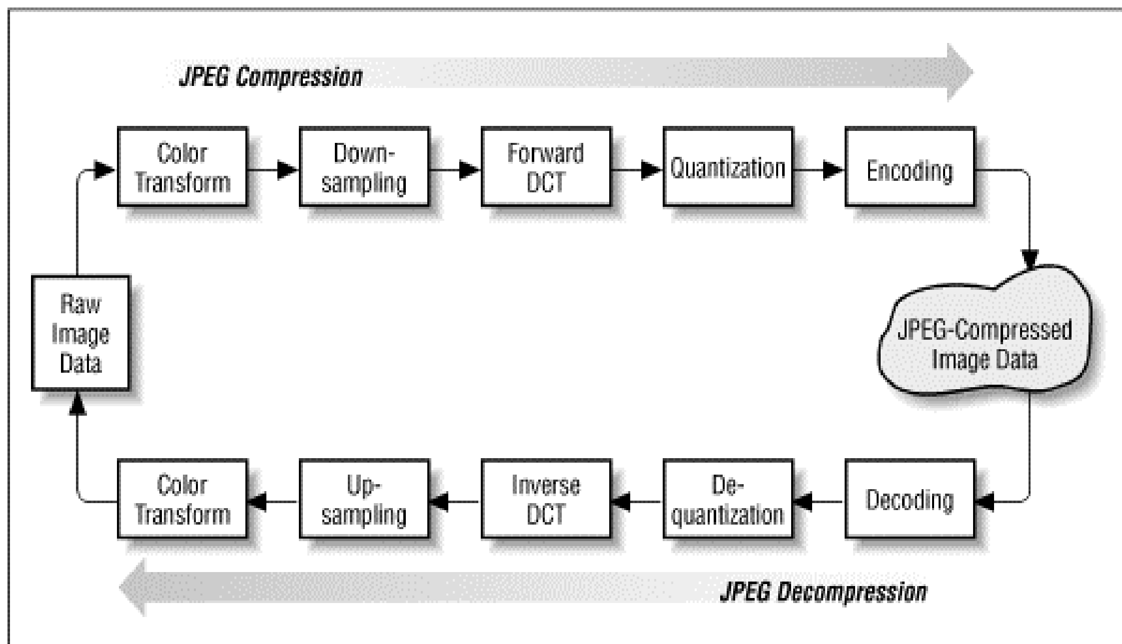


Bild 1: Baseline Code

2.2.1 Colortransformation & Colorrücktransformation

Bei der Colortransformation wird das **RGB-Modell** in das **YCbCr-Modell** umgerechnet. Das YCbCr-Modell ist ein Helligkeit-Farbigkeit-Modell. Ein Farbwert wird durch eine Grundhelligkeit (Y) und dessen Abweichung von Grau in Richtung Blau (Cb) und Rot (Cr) bestimmt. Die meiste Information liegt in der Grundhelligkeit, so dass man nur noch die Abweichungen nach Rot und Blau darzustellen braucht.

Bei der Colorrücktranformation rechnet man vom YCbCr-Modell zurück ins RGB-Modell.

$$\begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix} = \begin{bmatrix} 0.2989 & 0.5866 & 0.1145 \\ 0.5000 & -0.4183 & -0.0817 \\ -0.1688 & -0.3312 & 0.5000 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

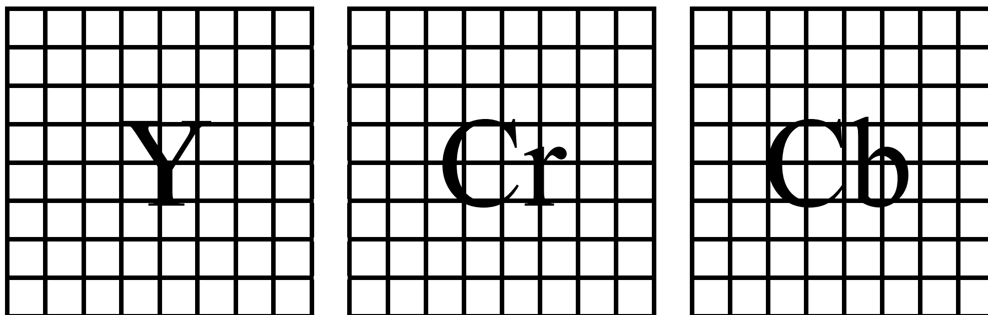
Formel 1: Colortransformations-Matrix

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1.4022 & 0.0001 \\ 1 & -0.7145 & -0.3458 \\ 1 & 0.0001 & 1.7710 \end{bmatrix} * \begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix}$$

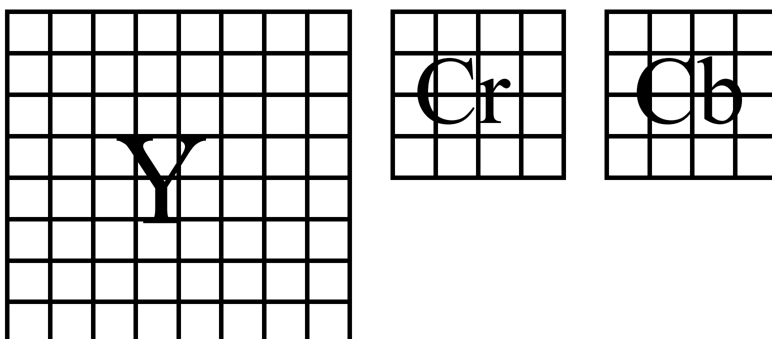
Formel 2: Colorrücktransformations-Matrix

2.2.2 Down- & Up-Sampling

Das Auge kann feine Unterschiede nebeneinander liegender Werte Cb bzw. Cr nur sehr schlecht wahrnehmen. Daher werden die Farbwerte für Bereiche von üblicherweise 2x2 Pixels gemittelt, so dass nicht der Farbwert jedes Punktes kodiert werden muss. Somit wird bereits die Hälfte aller Daten gespart.



64 Pixel + 64 Pixel + 64 Pixel = **192 Pixel**



64 Pixel + 16 Pixel + 16 Pixel = **96 Pixel**

Für das Upsampling werden die Werte einfach wieder vervielfacht.

2.2.3 ForwardDCT & InverseDCT

Eine Grafik wird bei der DCT in Blöcke zu je 8x8 Pixels aufgeteilt. Jedes Element in einem 8x8 Block wird mittels der Diskreten Cosinus Transformation nach folgenden Formeln transformiert und entsprechend mit der inversen DCT beim entkomprimieren zurücktransformiert (ohne die Formeln jetzt näher erläutern zu wollen):

$$\text{DCT}: \quad d_{mn} = \frac{1}{4} \cdot c_m \cdot c_n \cdot \sum_{m=0}^{7} \sum_{n=0}^{7} a_{ij} \cdot \cos \frac{(2 \cdot i + 1) \cdot m \cdot \pi}{16} \cdot \cos \frac{(2 \cdot j + 1) \cdot n \cdot \pi}{16}$$

$$\text{Inverse DCT}: a_{ij} = \frac{1}{4} \cdot \sum_{m=0}^{7} \sum_{n=0}^{7} c_m \cdot c_n \cdot d_{mn} \cdot \cos \frac{(2 \cdot i + 1) \cdot m \cdot \pi}{16} \cdot \cos \frac{(2 \cdot j + 1) \cdot n \cdot \pi}{16}$$

$$\text{mit} \quad c_m, c_n = \frac{1}{\sqrt{2}} \quad \text{für} \quad m, n = 0; \qquad \text{sonst} \quad c_m, c_n = 1$$

Formel 2: DCT und inverse DCT

Nach der Transformation entsteht wieder ein 8x8 Block. Das Element in der linken oberen Ecke entspricht nun dem DC-Anteil und die restlichen 63 Elemente sind AC-Anteile. Jedes AC-Element entspricht dem Anteil der Frequenz an welcher Position sich dieses Element befindet. Die Frequenzen nehmen von links nach rechts und von oben nach unten zu d.h. das Element rechts unten entspricht dem Anteil der höchsten Frequenz.

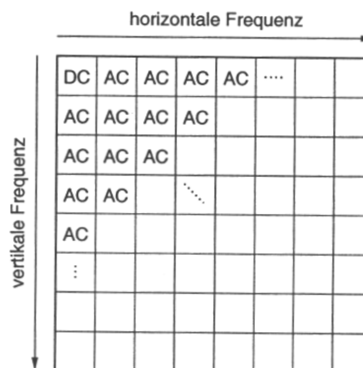


Bild 2:8x8 Block Positionen der DCT-Koeffizienten in Abhängigkeit von der Ortsfrequenz

Große regelmäßige Flächen im Bild schlagen sich in niedrigen Frequenzanteilen nieder, feine Details und genaue Auflösung von Farbunterschieden in hohen. Die DCT nutzt die Schwächen des menschlichen Auges und filtert die hohen Ortsfrequenzen heraus, die das Auge ohnehin nicht wahrnehmen kann. Da sich benachbarte Pixelwerte in der Regel kaum unterscheiden, werden nach der DCT nur der DC-Koeffizient und einige niederfrequente AC-Koeffizienten größere Werte annehmen.

2.2.4 Quantisierung & Dequantisierung

Auf die DCT folgt die Quantisierung. Es wird nicht für jeden Wert im 8x8 Block den gleichen Quantisierungfaktor benötigt. Da das menschliche Auge die hohen Frequenzanteile nicht allzu gut wahrnimmt, werden meist für die höheren Frequenzanteile besonders hohe Quantisierungsfaktoren benutzt. Die DCT-Koeffizienten werden durch den dazugehörenden Quantisierungsfaktor geteilt und auf den nächsten Integerwert gerundet. Dadurch dass bei hohen Frequenzen, hohe Quantisierungsfaktoren benutzt wurden, führt dies dazu, dass nach der Quantisierung viele Elemente den Wert Null aufweisen. Die Dequantisierung multipliziert den quantisierten Wert später einfach wieder mit dem Quantisierungsfaktor. Durch die dabei entstehenden Rundungsfehler gehen Informationen verloren. Bei JPEG kann man den Grad der Kompression wählen, dabei wird einfach nur der Quantisierungsfaktor entsprechend skaliert. Eine Kompression von < 1/10 ist ohne großen Informationsverlust möglich, zu starke Komprimierung führt allerdings zu Artefakten, das bedeutet, dass die Blockstruktur des Gesamtbildes sichtbar wird. Das Bild wirkt "pixelig". Beispiele für mögliche Quantisierungstabellen findet man in der Realisierung unter dem Punkt Quantisierung & Dequantisierung.

2.2.5 Kodierung & Dekodierung

Die vorgestellten Verfahren beinhalten noch keine direkte Kompression der Bilddaten, diese werden aber entsprechend (grob) transformiert und aufbereitet. Um die so erhaltenen Daten schließlich in einem möglichst kompakten Code abzuspeichern, stellt der JPEG-Standard mehrere Verfahren bereit:

- Komprimierung durch Huffman-Algorithmus
- Darstellung von variable-length-integers, anstatt Integers fester Länge
- Arithmetisches Codieren

Beim Huffman-Algorithmus kann man sich nun zu Nutze machen, dass man einen sehr geringen Anteil hoher Frequenzen hat. Schaut man sich nach der Quantisierung einen 8x8 Block an, sieht man, dass viele Werte Null sind. Nun macht man ein sogenanntes Zick-Zack-Scanning und kodiert die Werte von den tiefen Frequenzen zu den Hohen. Je höher die Frequenzen, umso wahrscheinlicher haben diese den Wert Null. Dies macht man sich nun zu Nutze und kodiert nur die AC-Wert ungleich Null. Dazu muss man jedoch wissen wie gross der Abstand zum Vorgänger ungleich Null ist. Diese Lauflänge darf einen Wert von 0 bis 15 haben. Da die Anzahl benachbarter Nullen und die Grösse des Nachfolgenden Koeffizienten miteinander korreliert sind, werden aus der Kombination von Lauflänge und Kategorie des AC-Koeffizienten neue Datensymbole gebildet. Dabei sind zwei besondere Symbole zu unterscheiden. ZRL definiert eine Lauflänge von 15 Nullen mit einem Folgewert Null. Das Ende des Zick-Zack-Scans wird durch EOB (**E**nd**O**f**B**lock) markiert. Meist braucht man über die Hälfte aller 64 Werte im 8x8 Block nicht zu kodieren resp. kann man zusammennehmen. Dies erspart eine Menge an Daten.

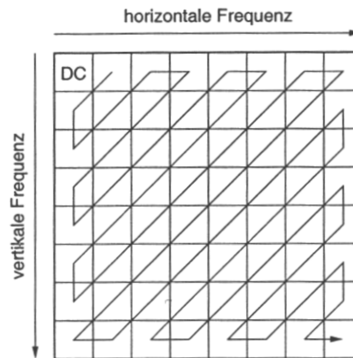


Bild 3: Zick-Zack-Scanning der Koeffizienten nach aufsteigenden Frequenzen

Den aus der Kombination entstandenen, neuen Datensymbolen werden Huffman-Code-Wörter zugeordnet. Einen Auszug für einen Code-Vorschlag kann man in der Realisierung unter Kodierung nachschlagen. Entsprechend der statistischen Verteilung werden bei gleicher Lauflänge kürzere Codes für kleinere AC-Werte und bei gleicher Kategorie kürzere Codes für kurze Lauflängen verwendet. Dem Huffman-Code folgen wiederum weitere Bits, die den tatsächlichen Koeffizientenwert festlegen müssen. Die Anzahl der Bits ist durch die Kategorienummer bestimmt.

Bei der Decodierung werden die Huffman-Codes wieder in DC- & AC-Werte decodiert. Es ist ganz einfach das umgekehrte Verfahren.

3 Beispiel

Im Folgenden werden einmal alle Schritte der Komprimierung und Dekomprimierung an einem 8x8Block gezeigt (für Y-Werte). Die Colortransformation resp. Colorrücktransformation und das Down-/Up-Sampling werden hier nicht gemacht.

In Bild 4:

(a) Der Block ist gefüllt mit möglichen Grauwerten
(b) Nach der DCT haben wir folgende Werte
(c) Hier sehen wir als Beispiel eine JPEG standardisierte Quantisierungstabelle für Luminanz die wir nun anwenden um die DCT Koeffizienten zu quantisieren.

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

(a) Source image samples

235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3
-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2
-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1
-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3
-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3
1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0
-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8
-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4

(b) forward DCT coefficients

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

(c) quantization table

Bild 4

In Bild 5:

(a) Nach der Quantisierung haben wir folgende Werte. Diese Werte werden nun kodiert. Zuerst wird der DC-Wert kodiert. Man ermittelt in welcher Kategorie er sich befindet. Anschliessend nimmt man in unserem Fall die DC-Huffman-Code-Tabelle für Luminanz (findet man unter dem Punkt Realisierung) und liesst den Code heraus. Das Gleiche macht man nun mit den AC-Werten. Hier muss man beachten das Erstens die AC-Werte in der Reihenfolge des Zick-Zack-Scannings kodiert werden und Zweitens die Lauflänge eine Rolle spielt.
Im Datenstrom wird immer zuerst der Code geschrieben und anschliessend der DC-Wert resp AC-Wert.
Für unseren Beispielblock sieht das folgendermassen aus.

Lauflänge/Kategorie	-/4	1/2	0/1	0/1	0/1	2/1
Codewort	1110 1111	11011 01	00 0	00 0	00 0	11100 0

(b) Die Werte werden quantisiert
(c) Hier sieht man die rekonstruierten Werte

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-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
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(a) normalized quantized coefficients

240	0	-10	0	0	0	0	0
-24	-12	0	0	0	0	0	0
-14	-13	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
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b) denormalized quantized coefficients

144	146	149	152	154	156	156	156
148	150	152	154	156	156	156	156
155	156	157	158	158	157	156	155
160	161	161	162	161	159	157	155
163	163	164	163	162	160	158	156
163	164	164	164	162	160	158	157
160	161	162	162	162	161	159	158
158	159	161	161	162	161	159	158

(c) reconstructed image samples

Bild 5

4 Realisierung

4.1 Wie arbeitet das vorliegende Client-Server Programm?

Das Programm ist in Visual C++ geschrieben worden, welches gleichzeitig unsere Entwicklungsumgebung ist. Das meiste wurde mit Hilfe der MFC (Microsoft Foundation Class Library) implementiert.
Der Teilnehmer, der eine Kommunikation anfängt, wird Client genannt. Da die Kommunikation unidirektional ist, behält der Teilnehmer, der an die Videokamera angeschlossen ist, auch diesen Status. Die Kamera schießt ein Bild und schreibt es auf das Clipboard, welches auf Windows bereitgestellt wird. Sobald das nächste Bild von der Camera kommt wird dieses überschrieben.
Vom Clipboard her ist es nun möglich die einzelnen Bilder durch Benutzung der MFC, in einen Buffer einzulesen. Dort sind nun die einzelnen Pixel in RGB-Werte vorhanden.

A_{11}	A_{12}											A_{1k}															A_{1n}
A_{21}	A_{22}											A_{2k}															A_{2n}
A_{i1}	A_{i2}											A_{ik}															A_{in}
A_{m1}	A_{m2}											A_{mk}															A_{mn}

Bild 4: Originalbild
A_{ik} : Pixel; i : Zeilenindex; k : Spaltenindex

<table>
<tr><td>B</td><td>G</td><td>R</td><td>B</td><td>G</td><td>R</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>B</td><td>G</td><td>R</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>B</td><td>G</td><td>R</td></tr>
<tr><td colspan="3">A_{11}</td><td colspan="3">A_{12}</td><td colspan="3"></td><td colspan="3"></td><td colspan="3"></td><td colspan="3"></td><td colspan="3">A_{ik}</td><td colspan="3"></td><td colspan="3"></td><td colspan="3"></td><td colspan="3"></td><td colspan="3"></td><td colspan="3"></td><td colspan="3">A_{mn}</td></tr>
</table>

Bild 5-1: Werte im Array bei 24-Bit

<table>
<tr><td>B</td><td>G</td><td>R</td><td>T</td><td>B</td><td>G</td><td>R</td><td>T</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>B</td><td>G</td><td>R</td><td>T</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>B</td><td>G</td><td>R</td><td>T</td></tr>
<tr><td colspan="4">A_{11}</td><td colspan="4">A_{12}</td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4">A_{ik}</td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4">A_{mn}</td></tr>
</table>

Bild 5-2: Werte im Array bei 32-Bit

Je nachdem ob Ihr Rechner eine Farb-Qualität von 24-Bit oder 32-Bit unterstützt, enthält Ihr Array noch ein Transparent-Bit, welches wir jedoch ignorieren.
Der Buffer enthält Charakter, welche die Werte unsigned, also von –128...127 speichert. Die einzelnen Pixel sind nach der oben Dargestellten Reihe im Buffer.
Mit dem gleichen Datentype werden die Bildinformationen nun einem UDP-Socket übergeben, welcher die Daten mittels Pakete über das Netzwert zum Server sendet.

Das vorliegende Server-Programm macht nichts anderes als die erhaltenen Pixelinformationen in richtiger Reihenfolge anzuzeigen.

4.2 Zugriff und Behandlung der einzelnen Elemente

Der Zugriff auf die Werte im eindimensionalen Buffer Array kann leicht durch Zeigerarithmetik implementiert werden. Es muss jedoch darauf geachtet werden, das der Zeiger nach Manipulationen wieder auf den Anfang des Buffers zeigt. Ansonsten erfolgen Zugriffe auf nicht freigegebene Speicherzellen.

4.2.1 Statisch erzeugte mehrdimensionale Matrizen

Eine erste Idee war es, die Werte in mehrere Matrizen zu werfen um eine einfache parallele Ansichtsweise zum Originalbild zu haben. So wurden die RGB Werte getrennt und in der jeweiligen Matrix an die Stelle kopiert, wo sie sich auch im Bild befinden.

Nachteilig war jedoch, dass die Werte durch den oben beschriebene Baseline Code sehr oft in neue Matrizen kopiert werden mussten, welches sehr lange Rechenzeiten benötigte.
Ein zweites Problem war in der Statik. Solche Matrizen, also mehrdimensionale Array, konnten nur mit Konstanten Werten dimensioniert werden. Das bedeutet, dass die grösse des Bildes schon beim Kompilieren festgelegt werden musste. Nicht besonders schön!

4.2.2 Dynamisch erzeugte mehrdimensionale Matrizen

Damit die Bildgrösse nicht schon zum Entwicklungszeitpunkt fixiert werden muss, wurden die Matrizen dynamisch, als mit „new“ erzeugt. Dabei musste man darauf achten, das man temporäre Matrizen mit „delete“ wieder frei gab, um keine memory leak zu erhalten. Alles andere konnte vom statischen Modell übernommen werden.
Es stellte sich jedoch heraus, dass so die Rechenzeit noch etwas länger wurde, denn das Herumkopieren der Werte fand weiterhin im gleichen Rahmen statt.

4.2.3 Matrix-Klasse

Der nächste Versuch war mit einer bereits existierende Matrix-Klasse, die man unter http://www.techsoftpl.com/matrix kostenlos herunterladen kann. Mit dieser Klasse wird das erzeugen, löschen und rechen mit Matrizen sehr einfach.
Für eine statische, also keine Echtzeitapplikation, währe diese Methode sicher eine gute Option und einfach zu implementieren. Da die Klasse aber viel mehr kann, als benötigt wird, muss man das mit enormer Rechenleistung bezahlen. Viel zu langsam für unser Problem!

4.2.4 Buffer und Zeiger

Die Idee hinter dieser Methode ist, die Werte so lange wie möglich im gleichen Format und am gleichen Ort zu halten. Das bedeut das die Originalwerte aus dem Buffer geholt, bearbeitet und wieder an die selbe Stelle geschrieben werden. Bei Berechnungen bei denen man von anderen Pixelwerten abhängig ist, braucht man jedoch noch temporäre Buffer, welches kein Problem ist, sofern man die in *Bild 3* gezeigte Struktur einhält.
Die Rechenzeit hielt sich in Grenzen, worauf diese Methode für die Colortransformation und das Down-Sampling beibehalten wurde, da nur diese Manipulationen mit allen Werten durchgeführt werden.
Für das weitere Vorgehen werden fixe Blöcke von 8x8 Werten verwendet. Diese Schritte können wir also problemlos mit statischen 8x8-dimensionalen Arrays realisieren.

4.2.5 Definition der Wertebereiche

Da nicht überall mit dem gleichen Wertebereich gearbeitet werden kann, sind die Bereiche wie folgt definiert worden:

Schritt	*Wertebereich*	*Datentype*
Einlesen	-128…127	char
Color-Transformation	0…255	float

Downsampling	0…255	float
DCT	$\pm 3.4*10^{-38}$ ... $\pm 3.4*10^{38}$	float
Quantisierung	$\pm 3.4*10^{-38}$ ... $\pm 3.4*10^{38}$	float
Encoding	-128...127	char

Tabelle 1: Wertebereiche

4.3 Colortransformation

Diese Transformation ist im wesentlichen nur eine Multiplikation der RGB-Werte eines Pixels mit der in 2.2.1 genanten Matrix.

B G R	B G R					B G R				B G R
A_{11}	A_{12}					A_{ik}				A_{mn}

⇓
Color-Transformation
⇓

Y Cr Cb	Y Cr Cb					Y Cr Cb					Y Cr Cb
A_{11}	A_{12}					A_{ik}					A_{mn}

Bild 4: Werte im Array

Jedoch muss man darauf achten, das diese Matrix die R/G/B/Y Werte im Bereich 0...255 und die Cr/Cb Werte im Bereich –128...127 benötigt, respektive errechnet. Es gibt jedoch verschiedene Matrizen für diese Transformation, welche die Wertebereiche wider anders Handhaben.
Bei der Rücktransformation muss man einfach die Inverse-Color-Transformations-Matrix benützten, der Rest bleibt gleich.

4.4 Downsampling

Ist nur eine Durchschnittsbildung von vier nebeneinanderliegenden Pixels, allerdings nur mit den Cr- und Cb-Werten. Hier wird eine erste wirkliche Datenreduktion von 50% gemacht, wobei aber Bildinformationen weggeworfen werden, die nicht mehr rekonstruierbar sind.

$$Cr_{mn} = \frac{Cr_{mn} + Cr_{m+1n} + Cr_{mn+1} + Cr_{m+1n+1}}{4}$$

Formel 3: Downsampling

$$\begin{bmatrix} Cr & Cr \\ Cr & Cr \end{bmatrix} \Rightarrow \begin{bmatrix} Cr \end{bmatrix} \qquad \begin{bmatrix} Cb & Cb \\ Cb & Cb \end{bmatrix} \Rightarrow \begin{bmatrix} Cb \end{bmatrix}$$

Bild 5: Downsampling im Originalbild

Y	**Cr**	**Cb**	Y	~~Cr~~	~~Cb~~	Y	~~Cr~~	~~Cb~~	Y	~~Cr~~	~~Cb~~	Y	Cr	Cb				Y	Cr	Cb				Y	~~Cr~~	~~Cb~~
A_{11}			A_{12}			A_{13}			A_{14}			A_{15}						A_{ik}						A_{mn}		

Bild 4: Werte im Array

Somit haben 50% der Werte in unserem Buffer keine Bedeutung mehr, für die restlichen Schritte.
Beim Up-Sampling werden dann die fehlenden Werte durch den Durchschnittswert ersetzt.

4.5 Werte-Aufbereitung für DCT

Die im Originalbild beieinanderliegenden Werte werden in 8x8 Blöcke geschrieben. Wie schon oben erwähnt, sind das zweidimensionale Arrays. Es sind jedoch nur die Y-Werte, die genau beieinander liegen. Von den Cr und Cb-Werten werden ja nur noch pro 2x2-Pixelblock einen Wert, den Durchschnittswert genommen.
So bekäme man zum Beispiel bei einem Bild von 16x16 Pixels, 4 Y-Blöcke, 1 Cr-Block und 1 Cb-Block. Mit der folgenden Formel lassen sich die Anzahl Blöcke für jedes Bild berechen.

$$\underbrace{\frac{Breite}{8}\cdot\frac{Höhe}{8}}_{\text{Anzahl Y-Blöcke}}+\underbrace{\frac{Breite}{16}\cdot\frac{Höhe}{16}}_{\text{Anzahl Cr-Blöcke}}+\underbrace{\frac{Breite}{16}\cdot\frac{Höhe}{16}}_{\text{Anzahl Cb-Blöcke}}=\text{Totale Anzahl 8x8 Blöcke}$$

Formel 4: Blockberechnung

Die DCT wird nun in eine for-Schleife eingepackt, die einen 8x8 Block nach dem anderen übergibt.

4.6 Diskrete Cosinus Transformation

Bei der Realisierung der DCT-Formel in C++ hat man eigentlich keine grossen Probleme. Ich habe die Formel schön auseinandergenommen und analysiert. Man sieht schnell, dass es mit vier for-Schlaufen eigentlich funktioniert. Die Berechnung habe ich jeweils in den zwei tiefsten Schlaufen gemacht. Mit Hilfe von Zwischenwerten kann man noch Rechenzeit sparen.

$$X[0][0] = \frac{\frac{1}{\sqrt{2}}+\frac{1}{\sqrt{2}}}{4} * \left[\begin{array}{l} (x=0)\left\{\begin{array}{l} (y=0) \Rightarrow x[0][0]*\cos\left(\frac{1*0*\pi}{16}\right)*\cos\left(\frac{1*0*\pi}{16}\right)+ \\ (y=1) \Rightarrow x[0][1]*\cos\left(\frac{1*0*\pi}{16}\right)*\cos\left(\frac{3*0*\pi}{16}\right)+ \\ \ldots \\ (y=7) \Rightarrow x[0][7]*\cos\left(\frac{1*0*\pi}{16}\right)*\cos\left(\frac{15*0*\pi}{16}\right)+ \end{array}\right. \\ (x=1)\left\{\begin{array}{l} (y=0) \Rightarrow x[1][0]*\cos\left(\frac{3*0*\pi}{16}\right)*\cos\left(\frac{1*0*\pi}{16}\right)+ \\ (y=1) \Rightarrow x[1][1]*\cos\left(\frac{3*0*\pi}{16}\right)*\cos\left(\frac{3*0*\pi}{16}\right)+ \\ \ldots \\ (y=7) \Rightarrow x[1][7]*\cos\left(\frac{3*0*\pi}{16}\right)*\cos\left(\frac{15*0*\pi}{16}\right)+ \end{array}\right. \\ (\ldots)\{ \\ (x=7)\{ \end{array}\right]$$

$$X[0][1] =$$

...
X[0][7] =
...
X[7][0..7]

Das Problem ist jedoch, dass durch diese direkte Implementation sehr viele Operationen benötigt werden. Durch eine schnelle Implementation der Diskreten Fourier-Transformation (DFT) kann man dieses Problem aber lösen. Wir benutzten neu eine Transformation Radix 4 FFT(FastFourierTransformation). Man findet diese auf http://momonga.t.u-tokyo.ac.jp/~ooura/fft.html . Das Ergebnis ist das gleiche wie bei einer normalen DCT jedoch viel schneller.
Bei der inversen DCT kann man genau gleich vorgehen.

Wenn man nun das zweidimensionale Array mit den DCT-Koeffizienten hat kann man zum nächsten Schritt der Quantisierung gehen.

4.7 Quantisierung

Wie schon oben beschrieben, wird jeder einzelne Wert anders quantisiert. Dies Realisiert man indem man zu unserem 8x8 Block (zweidimensionales Array) in dem die DCT-Werte stehen auch einen 8x8 Quantisierungsblock (zweidimensionales Array) macht. Durch dividieren der Elemente an der gleichen Stelle im Block erhält man einen neuen Block der Quantisiert wurde. Standartmässig benutzt man zwei verschieden Quantisierungsblöcke für Luminanz und Chrominanzen. Diese Quantisierungswerte basieren auf der Tatsache, dass das menschliche Auge nicht alle Frequenzen und alle Farben gleich gut sieht. Folgend ist ein Beispiel für zwei Quantisierungstabellen.

Luminanz

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	72	95	98	112	100	103	99

Chrominanzen

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Nach der Quantisierung muss man eigentlich nur noch die erhaltenen Werte kodieren.

4.8 Kodierung

Wie schon beschrieben werden die DC- und AC-Werte mittels Huffman-Code-Tabelle kodiert. JPEG hat verschiedene Huffman-Code-Tabellen standardisiert welche auf Erfahrungswerten basiert und optimiert sind. Nachfolgend sind Tabellen für die Kodierung von DC-Werten und AC-Werte aufgeführt.
Zum weiteren Kodieren braucht man neben dem Code noch die jeweilige Bitlänge. Mit diesen Angaben kann man nun einen Datenstrom erzeugen. In unserem Fall haben wir die Daten in char verpackt. Einen Lösungsansatz für das Verpacken der Daten findet man in der Source(client.cpp). Grundsätzlich haben wir mit einer switch-Anweisung gearbeiten und somit Code und Länge mitgegeben.

4.9 Dekodierung

Hier haben wir mit Hilfe einer Matrix gearbeitet. Alle Datensymbole sind negativ enthalten. Die positiven Zahlen einsprechen einer Zeilennummer innerhalb der Matrix. Man durchläuft nun die Matrix Bit für Bit des übermittelten Datenstroms, bis man auf einen negativen Wert innerhalb der Matrix stösst. Sobald man somit ein Datensymbol erkennt hat, weiss man den Kategoriewert und bei AC-Werten zusätzlich noch die Lauflänge. Somit weiss man in wie vielen Bits nun der Wert folgt und wo im Block er hineingeschrieben werden muss. Im Anhang zur Kodierung & Dekodierung findet man die Matrizen für die Decodierung der DC-Werte der Lunminanz und Chrominanz. Für die AC-Wert findet man die Matrizen in der Source(server.cpp).

4.10 Anhang: Kodierung & Dekodierung

Einteilung der DC-Werte in Kategorien

Category	DC-Wert
0	0
1	-1,1
2	-3,-2,2,3
3	-7,...,-4,4,...,7
4	-15,...,-8,8,...,15
5	-31,...,-16,16,...,31
6	-63,...,-32,32,...,63
7	-127,...,-64,64,...,127
8	-255,...,-128,128,...,255
9	-511,...,-256,256,...,511
10	-1023,...,-512,512,...,1023
11	-2047,...,-1024,1024,...,2047

Einteilung der AC-Werte in Kategorien

Category	AC-Wert
0	0
1	-1,1
2	-3,-2,2,3
3	-7,...,-4,4,...,7
4	-15,...,-8,8,...,15
5	-31,...,-16,16,...,31
6	-63,...,-32,32,...,63
7	-127,...,-64,64,...,127
8	-255,...,-128,128,...,255
9	-511,...,-256,256,...,511
10	-1023,...,-512,512,...,1023

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Anhang 1: DC-Wert Huffman-Code-Tabelle für Luminanz

Category	Code length	Code word
0	2	00
1	2	01
2	2	10
3	3	110
4	4	1110
5	5	11110
6	6	111110
7	7	1111110
8	8	11111110
9	9	111111110
10	10	1111111110
11	11	11111111110

Anhang 2: DC-Wert Huffman-Code-Tabelle für Chrominanz

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110
3/1	6	111010
3/2	9	111110111
3/3	12	111111110101
3/4	16	1111111110001111
3/5	16	1111111110010000
3/6	16	1111111110010001
3/7	16	1111111110010010
3/8	16	1111111110010011
3/9	16	1111111110010100
3/A	16	1111111110010101

Run/Size	Code length	Code word
4/1	6	111011
4/2	10	1111111000
4/3	16	1111111110010110
4/4	16	1111111110010111
4/5	16	1111111110011000
4/6	16	1111111110011001
4/7	16	1111111110011010
4/8	16	1111111110011011
4/9	16	1111111110011100
4/A	16	1111111110011101
5/1	7	1111010
5/2	11	11111110111
5/3	16	1111111110011110
5/4	16	1111111110011111
5/5	16	1111111110100000
5/6	16	1111111110100001
5/7	16	1111111110100010
5/8	16	1111111110100011
5/9	16	1111111110100100
5/A	16	1111111110100101
6/1	7	1111011
6/2	12	111111110110
6/3	16	1111111110100110
6/4	16	1111111110100111
6/5	16	1111111110101000
6/6	16	1111111110101001
6/7	16	1111111110101010
6/8	16	1111111110101011
6/9	16	1111111110101100
6/A	16	1111111110101101
7/1	8	11111010
7/2	12	111111110111
7/3	16	1111111110101110
7/4	16	1111111110101111
7/5	16	1111111110110000
7/6	16	1111111110110001
7/7	16	1111111110110010
7/8	16	1111111110110011
7/9	16	1111111110110100
7/A	16	1111111110110101
8/1	9	111111000
8/2	15	111111111000000

Run/Size	Code length	Code word
8/3	16	1111111110110110
8/4	16	1111111110110111
8/5	16	1111111110111000
8/6	16	1111111110111001
8/7	16	1111111110111010
8/8	16	1111111110111011
8/9	16	1111111110111100
8/A	16	1111111110111101
9/1	9	111111001
9/2	16	1111111110111110
9/3	16	1111111110111111
9/4	16	1111111111000000
9/5	16	1111111111000001
9/6	16	1111111111000010
9/7	16	1111111111000011
9/8	16	1111111111000100
9/9	16	1111111111000101
9/A	16	1111111111000110
A/1	9	111111010
A/2	16	1111111111000111
A/3	16	1111111111001000
A/4	16	1111111111001001
A/5	16	1111111111001010
A/6	16	1111111111001011
A/7	16	1111111111001100
A/8	16	1111111111001101
A/9	16	1111111111001110
A/A	16	1111111111001111
B/1	10	1111111001
B/2	16	1111111111010000
B/3	16	1111111111010001
B/4	16	1111111111010010
B/5	16	1111111111010011
B/6	16	1111111111010100
B/7	16	1111111111010101
B/8	16	1111111111010110
B/9	16	1111111111010111
B/A	16	1111111111011000
C/1	10	1111111010
C/2	16	1111111111011001
C/3	16	1111111111011010
C/4	16	1111111111011011

Run/Size	Code length	Code word
C/5	16	1111111111011100
C/6	16	1111111111011101
C/7	16	1111111111011110
C/8	16	1111111111011111
C/9	16	1111111111100000
C/A	16	1111111111100001
D/1	11	11111111000
D/2	16	1111111111100010
D/3	16	1111111111100011
D/4	16	1111111111100100
D/5	16	1111111111100101
D/6	16	1111111111100110
D/7	16	1111111111100111
D/8	16	1111111111101000
D/9	16	1111111111101001
D/A	16	1111111111101010
E/1	16	1111111111101011
E/2	16	1111111111101100
E/3	16	1111111111101101
E/4	16	1111111111101110
E/5	16	1111111111101111
E/6	16	1111111111110000
E/7	16	1111111111110001
E/8	16	1111111111110010
E/9	16	1111111111110011
E/A	16	1111111111110100
F/0 (ZRL)	11	11111111001
F/1	16	1111111111110101
F/2	16	1111111111110110
F/3	16	1111111111110111
F/4	16	1111111111111000
F/5	16	1111111111111001
F/6	16	1111111111111010
F/7	16	1111111111111011
F/8	16	1111111111111100
F/9	16	1111111111111101
F/A	16	1111111111111110

Anhang 3: AC-Wert Huffman-Code-Tabelle für Luminanz

Run/Size	Code length	Code word
0/0 (EOB)	2	00
0/1	2	01
0/2	3	100
0/3	4	1010
0/4	5	11000
0/5	5	11001
0/6	6	111000
0/7	7	1111000
0/8	9	111110100
0/9	10	1111110110
0/A	12	111111110100
1/1	4	1011
1/2	6	111001
1/3	8	11110110
1/4	9	111110101
1/5	11	11111110110
1/6	12	111111110101
1/7	16	1111111110001000
1/8	16	1111111110001001
1/9	16	1111111110001010
1/A	16	1111111110001011
2/1	5	11010
2/2	8	11110111
2/3	10	1111110111
2/4	12	111111110110
2/5	15	111111111000010
2/6	16	1111111110001100
2/7	16	1111111110001101
2/8	16	1111111110001110
2/9	16	1111111110001111
2/A	16	1111111110010000
3/1	5	11011
3/2	8	11111000
3/3	10	1111111000
3/4	12	111111110111
3/5	16	1111111110010001
3/6	16	1111111110010010
3/7	16	1111111110010011
3/8	16	1111111110010100
3/9	16	1111111110010101
3/A	16	1111111110010110
4/1	6	111010

Run/Size	Code length	Code word
4/2	9	111110110
4/3	16	1111111110010111
4/4	16	1111111110011000
4/5	16	1111111110011001
4/6	16	1111111110011010
4/7	16	1111111110011011
4/8	16	1111111110011100
4/9	16	1111111110011101
4/A	16	1111111110011110
5/1	6	111011
5/2	10	1111111001
5/3	16	1111111110011111
5/4	16	1111111110100000
5/5	16	1111111110100001
5/6	16	1111111110100010
5/7	16	1111111110100011
5/8	16	1111111110100100
5/9	16	1111111110100101
5/A	16	1111111110100110
6/1	7	1111001
6/2	11	11111110111
6/3	16	1111111110100111
6/4	16	1111111110101000
6/5	16	1111111110101001
6/6	16	1111111110101010
6/7	16	1111111110101011
6/8	16	1111111110101100
6/9	16	1111111110101101
6/A	16	1111111110101110
7/1	7	1111010
7/2	11	11111111000
7/3	16	1111111110101111
7/4	16	1111111110110000
7/5	16	1111111110110001
7/6	16	1111111110110010
7/7	16	1111111110110011
7/8	16	1111111110110100
7/9	16	1111111110110101
7/A	16	1111111110110110
8/1	8	11111001
8/2	16	1111111110110111
8/3	16	1111111110111000

Run/Size	Code length	Code word
8/4	16	1111111110111001
8/5	16	1111111110111010
8/6	16	1111111110111011
8/7	16	1111111110111100
8/8	16	1111111110111101
8/9	16	1111111110111110
8/A	16	1111111110111111
9/1	9	111110111
9/2	16	1111111111000000
9/3	16	1111111111000001
9/4	16	1111111111000010
9/5	16	1111111111000011
9/6	16	1111111111000100
9/7	16	1111111111000101
9/8	16	1111111111000110
9/9	16	1111111111000111
9/A	16	1111111111001000
A/1	9	111111000
A/2	16	1111111111001001
A/3	16	1111111111001010
A/4	16	1111111111001011
A/5	16	1111111111001100
A/6	16	1111111111001101
A/7	16	1111111111001110
A/8	16	1111111111001111
A/9	16	1111111111010000
A/A	16	1111111111010001
B/1	9	111111001
B/2	16	1111111111010010
B/3	16	1111111111010011
B/4	16	1111111111010100
B/5	16	1111111111010101
B/6	16	1111111111010110
B/7	16	1111111111010111
B/8	16	1111111111011000
B/9	16	1111111111011001
B/A	16	1111111111011010
C/1	9	111111010
C/2	16	1111111111011011
C/3	16	1111111111011100
C/4	16	1111111111011101
C/5	16	1111111111011110

Run/Size	Code length	Code word
C/6	16	1111111111011111
C/7	16	1111111111100000
C/8	16	1111111111100001
C/9	16	1111111111100010
C/A	16	1111111111100011
D/1	11	11111111001
D/2	16	1111111111100100
D/3	16	1111111111100101
D/4	16	1111111111100110
D/5	16	1111111111100111
D/6	16	1111111111101000
D/7	16	1111111111101001
D/8	16	1111111111101010
D/9	16	1111111111101011
D/A	16	1111111111101100
E/1	14	11111111100000
E/2	16	1111111111101101
E/3	16	1111111111101110
E/4	16	1111111111101111
E/5	16	1111111111110000
E/6	16	1111111111110001
E/7	16	1111111111110010
E/8	16	1111111111110011
E/9	16	1111111111110100
E/A	16	1111111111110101
F/0 (ZRL)	10	1111111010
F/1	15	111111111000011
F/2	16	1111111111110110
F/3	16	1111111111110111
F/4	16	1111111111111000
F/5	16	1111111111111001
F/6	16	1111111111111010
F/7	16	1111111111111011
F/8	16	1111111111111100
F/9	16	1111111111111101
F/A	16	1111111111111110

Anhang 4: AC-Wert Huffman-Code-Tabelle für Chrominanz

Matrix zur Dekodierung der DC-Wert in Luminanz

$$M = \begin{bmatrix} 1 & 3 \\ 0 & 2 \\ -1 & -2 \\ 4 & 5 \\ -3 & -4 \\ -5 & 6 \\ -6 & 7 \\ -7 & 8 \\ -8 & 9 \\ -9 & 10 \\ -10 & 11 \\ -11 & 0 \end{bmatrix}$$

Matrix zur Dekodierung der DC-Wert in Chrominanz

$$M = \begin{bmatrix} 1 & 2 \\ 0 & -1 \\ -2 & 3 \\ -3 & 4 \\ -4 & 5 \\ -5 & 6 \\ -6 & 7 \\ -7 & 8 \\ -8 & 9 \\ -9 & 10 \\ -10 & 11 \\ -11 & 0 \end{bmatrix}$$

5 Resultat

5.1 Messresultate

In Windows Task Manager findet man eine einfache grafische Netzwerküberwachung, die den Datenfluss durch die eingestellte Netzwerkkarte aufzeichnet.
Im ersten Fenster sehen wir die Kurve des Originalen Client-Server Programme, die während 2 min aufgenommen wurde. Die Datenrate ist ziemlich konstant (Siehe Nr.1 Bild 1), da die Daten vom Programm direkt von der Kamera auf das Netzwert gesendet werden.

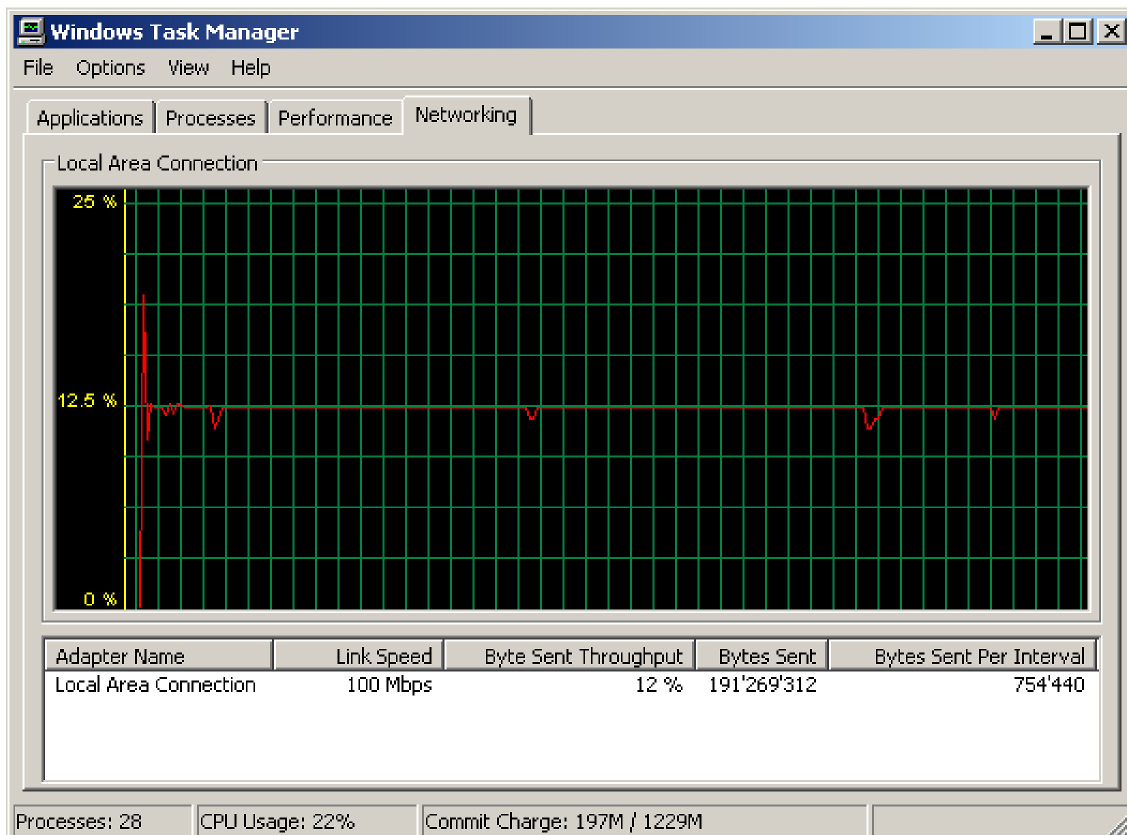


Bild 1: Bitrate Original

Im nächsten Fenster finden wir den Verlauf der Bitrate im modifizierten Client-Server Programm. Wird stellen fest, dass die Bitrate von durchschnittlich 12.5% auf 0.75%, das heisst etwa von 12.5Mbps auf 0.75Mbps gesunken ist. Das ergibt einen Kompressionsfaktor von 15-20. Haben wir jedoch ein sehr monotones Bild, das heisst fast keine hohen Frequenzen kann der Kompressionsfaktor bis auf 200 ansteigen (Siehe Nr. 1 in Bild 2). Dies wird vor allem erreicht durch die Lauflängenkodierung, da ausser dem DC-Wert fast alle Werte Null sind. Haben wir Standbilder so ist die Bitrate mehr oder weniger konstant, da immer wieder die gleichen Daten gesendet werden (Nr.2 in Bild 2). Änderns sich die Bilder vor der Kamera jedoch fortlaufen, schwankt die Bitrate dauernd (Nr.3 Bild2).

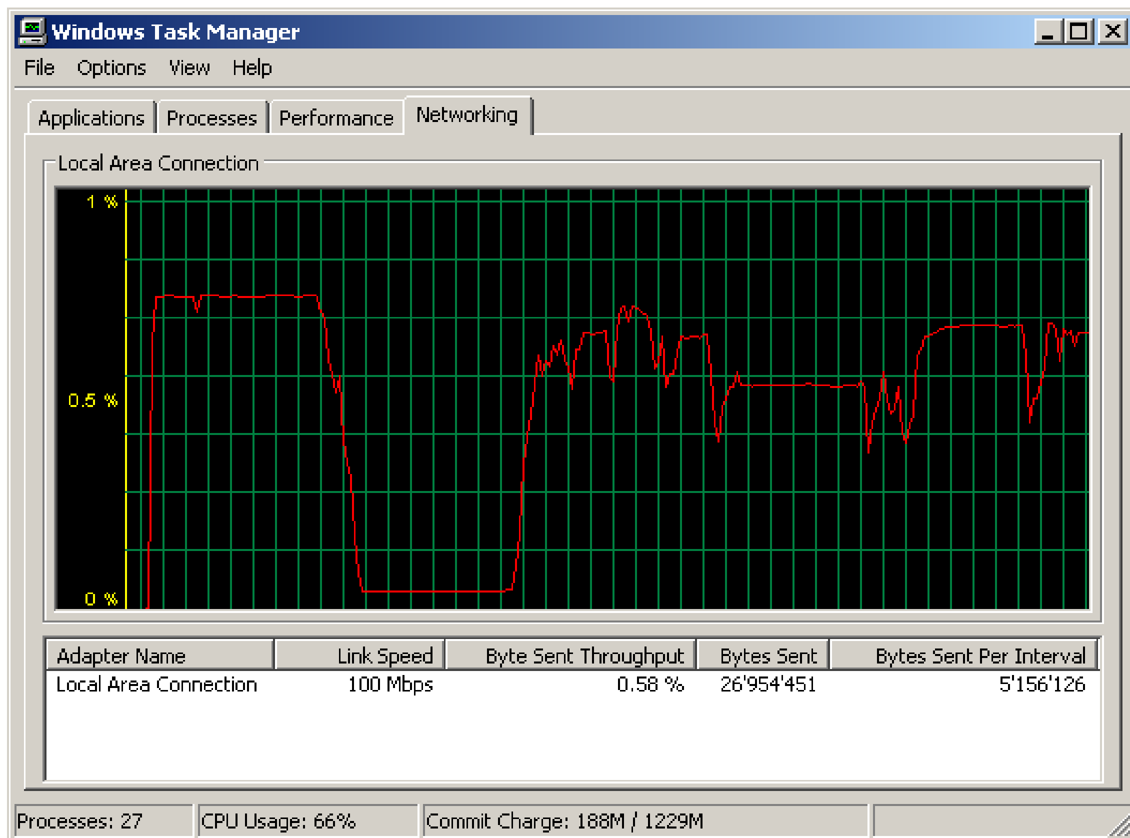


Bild 2: Bitrate modifiziert

5.2 Programm-Beschreibung

5.2.1 Mögliche Daten-Quellen

Das Programm unterstützt mindestens folgende Kameras und Formate:

- Fire-i400 1394 DIGITAL CAMERA unibran
  - 160 x 120 Pixel
  - 320 x 240 Pixel
- 3Com HomeConnect PC Digital Web Cam
  - 128 x 96 Pixel
  - 160 x 120 Pixel
  - 176 x 144 Pixel
  - 320 x 160 Pixel

Die Bildgröße ist jeweils beim Anwenderprogramm der Kamera anzugeben. Unser Video-Streamer passt sich dann automatisch dem Format an.

5.2.2 Client

Beim Client ist per Default die IP-Adresse des eigenen Rechners eingegeben, die die Kommunikation von Client und Server auf demselben Computer ermöglicht. Die Adresse kann jedoch dauernd geändert werden. Allerdings muss dafür der Standbild-Button (Siehe Nr.1 Bild 1) aktiviert werden um keinen aktiven Strom zu unterbrechen. Durch Eingabe in das

IP-Address-Field (Siehe Nr.2 Bild 1), kann danach die IP-Adresse des Rechners auf dem der Server läuft angegeben werden. Die Adresse wird beim drücken des Start-Button (Siehe Nr.3 Bild) aktualisiert und die Daten werden von nun an per UDP an die neue Adresse gesendet. Um das Programm zu beenden ist natürlich der Abbrech-Button zuständig (Siehe Nr.4 Bild 1).

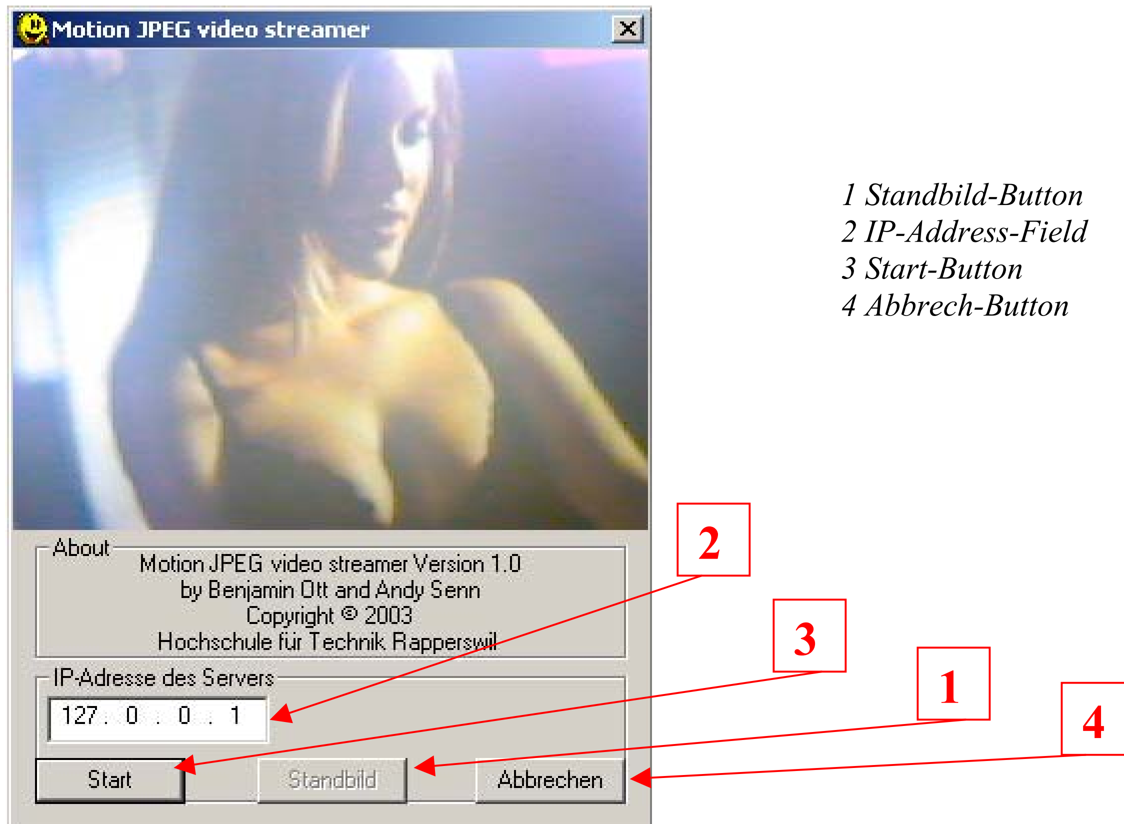


Bild 1: Client

5.2.3 Server

Beim Server funktionieren die Buttons gleich wie beim Client. Speziell ist hier die Eingabe der Farbqualität. Man hat die Auswahl zwischen 24-Bit und 32-Bit. Änderungen können bei Aktivierung des Standbild-Buttons gemacht werden. Per Default ist das Programm auf 32-Bit eingestellt, haben Sie einen solchen Rechner startet der Server bei betätigen des Start-Buttons sofort.
Das Format des Bildes wird automatisch angepasst.

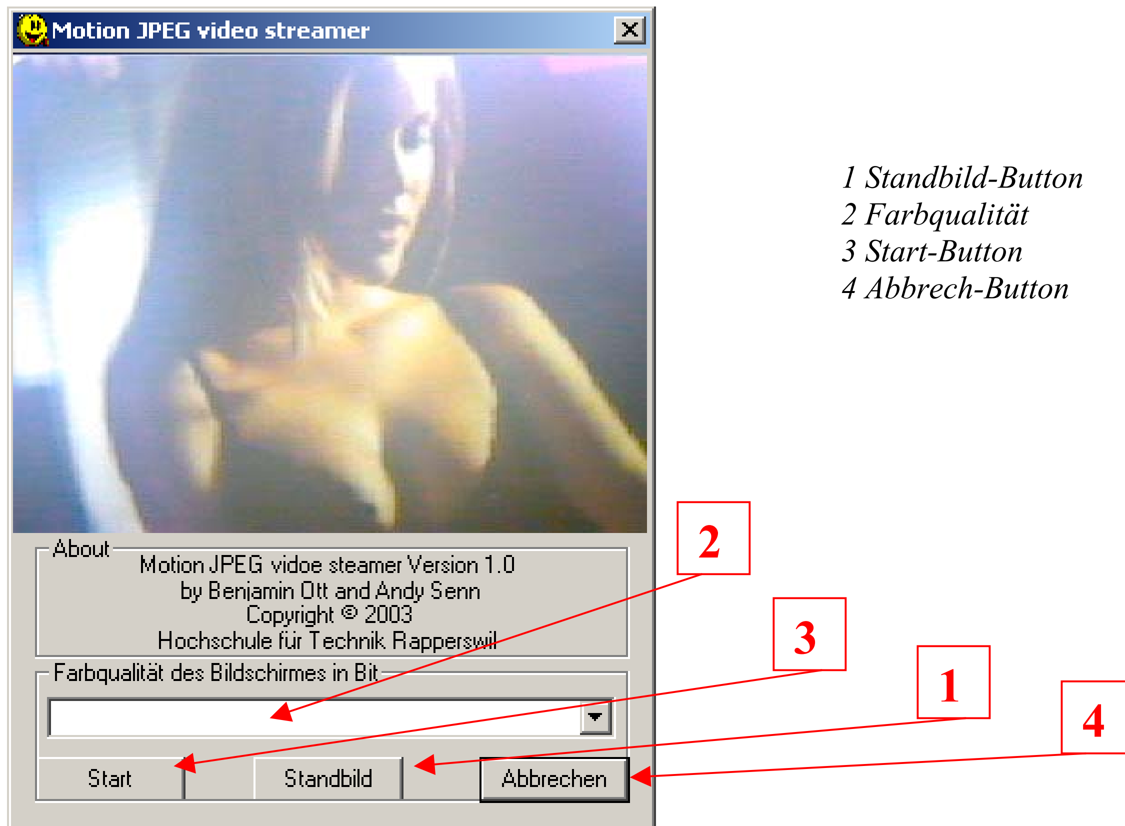


Bild 1: Server

5.3 Fazit

Das Programm läuft stabil, hat ein akzeptablen Komprimierungsfaktor und ist einfach zu bedienen. Jedoch brauch es relativ viel Rechenaufwand bei grösseren Bildern. Für Computer mit langsamen Prozessoren, führt dies zu stotternden Bildübergängen.