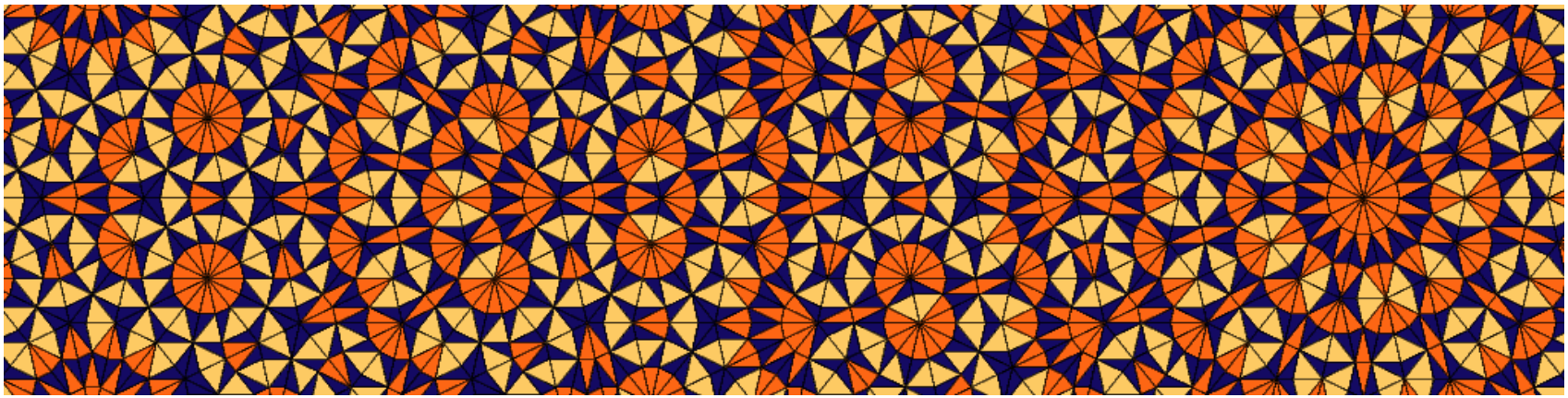
Panorama der Mathematik und Informatik

21: Algorithmen IV: png und Co

Dirk Frettlöh
Technische Fakultät / Richtig Einsteigen



Kompressionsalgorithmen:

Idee: Speichere 2 MB Daten in einer 1 MB Datei. Ist, je nach Daten, einfach bis sehr, sehr schwierig. Einfaches Bsp.:

00000000000000000011111111111111111100000000000011111000000000

Kniffliges Bsp.:

47238905619069345682390129012012356890317329045290239230237851

Kompressionsalgorithmen:

Idee: Speichere 2 MB Daten in einer 1 MB Datei. Ist, je nach Daten, einfach bis sehr, sehr schwierig. Einfaches Bsp.:

000000000000000000111111111111111111000000000000111110000000000

Kniffliges Bsp.:

472389056190693456823901290120123568903173290452902392302378 51

Häufig genutzte Verfahren:

- Run length encoding: (RLE) Blocklängen nutzen
- Huffman coding: häufige Zeichen mit wenig Bits (s.u.)
- LZ77, LZ78, LZW: Wiederholungen notieren (Wörterbuch anlegen, s.u.)

LZ77: Lempel-Ziv 1977. Nachfolger: LZ78, LZW (darauf zeitweise Patente, daher oft das freie LZ77).

Beispiel Run Length Encoding

000000000000000001111111111111111100000000000111110000000

Bsp. RLE: String oben speichern als:

18 0en, 18 1en, 12 0en, 5 1en, 9 0en.

Also etwa: (18,0,18,1,12,0,5,1,9,0). 10 statt 62 Zahlen.

Grenzen

Komprimieren (also “kleiner machen”) klappt nur, wenn die Daten irgendeine Regelmäßigkeit aufweisen.

Komplett zufällige Daten (“hohe Entropie”, siehe *Informationstheorie*) lassen sich nur schwer (oder gar nicht) komprimieren.

Anwendungen

DEFLATE: in `gzip`, WinZIP, MacOS Archivmanager... kombiniert LZ77 und Huffman coding.

(`zip` ist ein Dateiformat zum Packen mehrerer Dateien, das verschiedene Kompressionsmethoden benutzt, gzip packt nur eine Datei, und immer mit DEFLATE)

Anwendungen

DEFLATE: in `gzip`, WinZIP, MacOS Archivmanager... kombiniert LZ77 und Huffman coding.

(`zip` ist ein Dateiformat zum Packen mehrerer Dateien, das verschiedene Kompressionsmethoden benutzt, gzip packt nur eine Datei, und immer mit DEFLATE)

- Grafik: jpeg, gif, png... auch in pdf.
- Audio: mp3, ogg, MPEG-4, ...
- Video: flv, mov, MPEG-4, ...

Hier: Details zu jpeg und png (gif ist sehr ähnlich zu png)

Ein naives Grafikformat wäre "Pixel 1 hat Farbe #e34d76, Pixel 2 hat Farbe #ea4d73, Pixel 3 hat Farbe #f14e69, ..."

Anwendungen

DEFLATE: in `gzip`, WinZIP, MacOS Archivmanager... kombiniert LZ77 und Huffman coding.
(`zip` ist ein Dateiformat zum Packen mehrerer Dateien, das verschiedene Kompressionsmethoden benutzt, gzip packt nur eine Datei, und immer mit DEFLATE)

- Grafik: jpeg, gif, png... auch in pdf.
- Audio: mp3, ogg, MPEG-4, ...
- Video: flv, mov, MPEG-4, ...

Hier: Details zu jpeg und png (gif ist sehr ähnlich zu png)

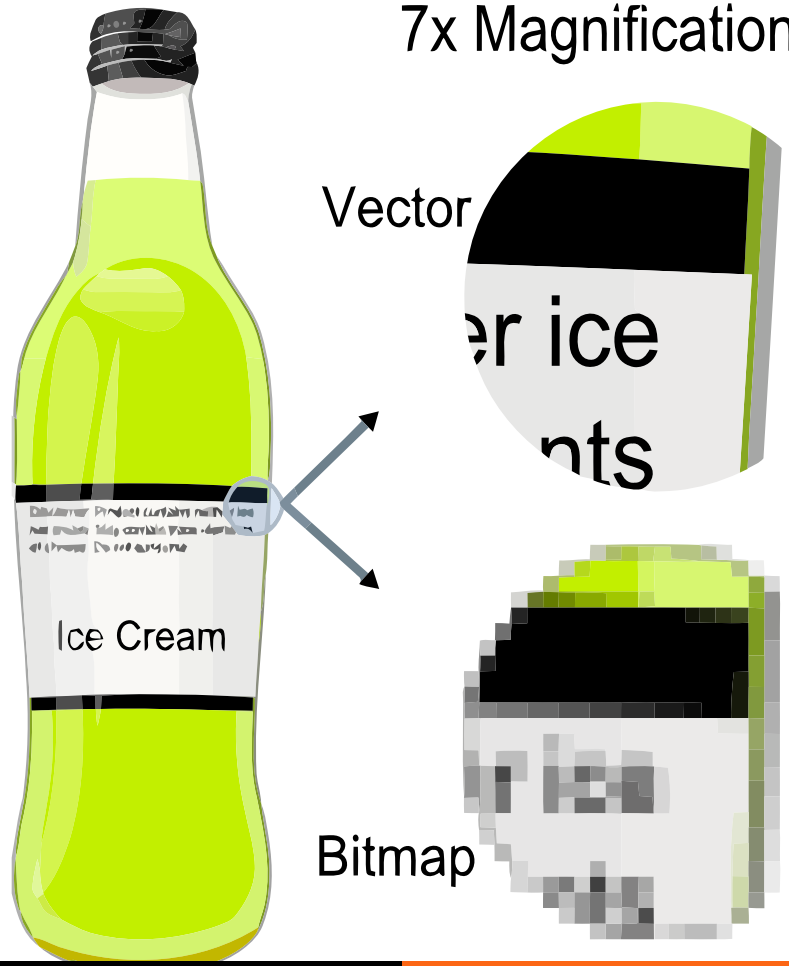
Ein naives Grafikformat wäre "Pixel 1 hat Farbe #e34d76, Pixel 2 hat Farbe #ea4d73, Pixel 3 hat Farbe #f14e69, ..."

Bekannt sein sollte: Farben werden digital als RGB (Rot-Grün-Blau) oder CMYK (cyan-magenta-yellow-black) gespeichert. Z.B. 8 bits für Rotanteil (0=00=wenig, 255=ff=knallrot), 8 bits für Grün, 8 bits für Blau. Z.B. # ff 00 00, # fd d7 4b

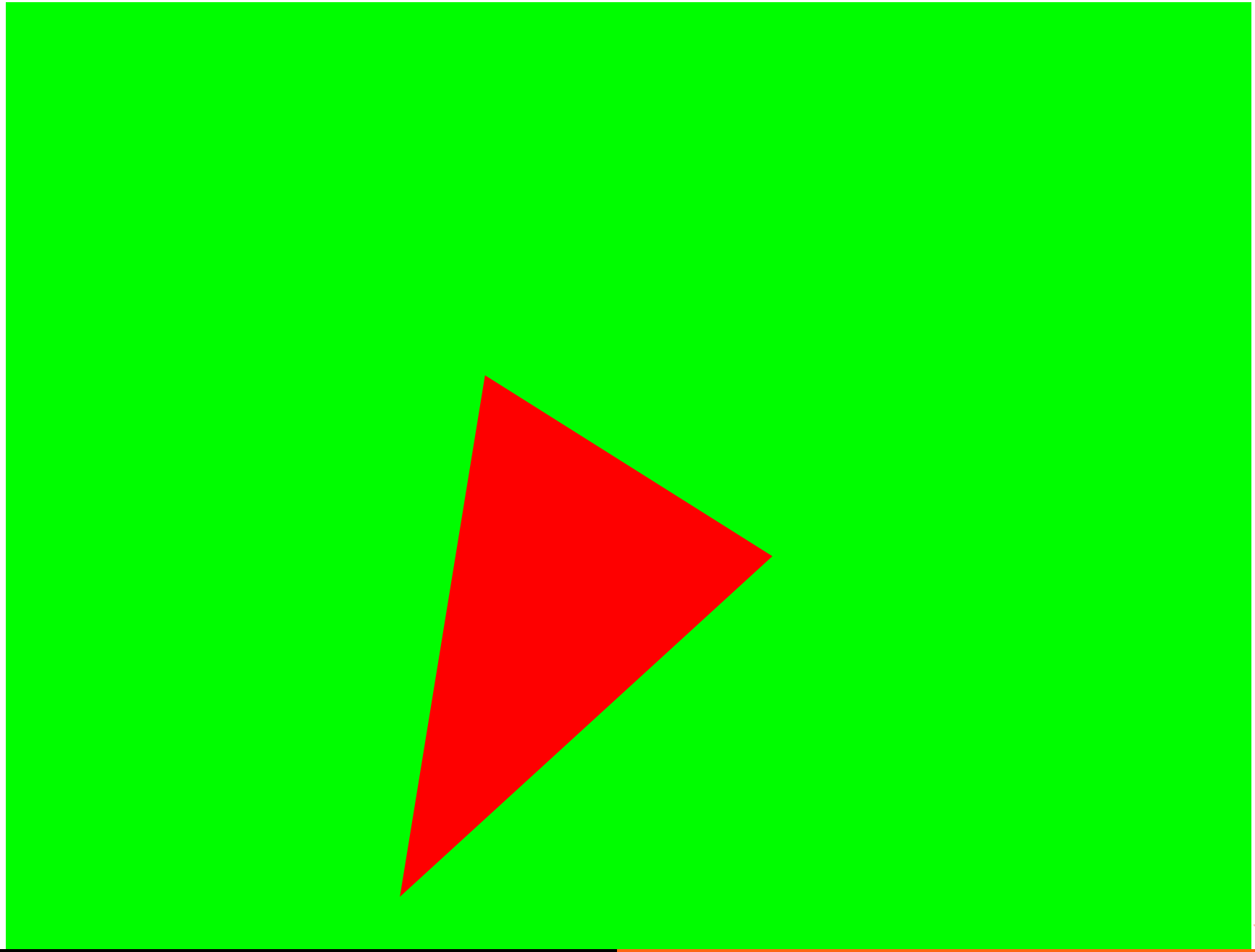
Rastergraphik vs Vektorgraphik

Rastergraphik: (engl. Bitmap): Bild besteht aus Pixeln. jpeg, png, gif, bmp, tiff...

Vektorgrafik nicht pixelbasiert. svg, ps, eps, pdf, ai, cdr, tikz....



Ein einfaches Bild, als pdf gespeichert. Reinzoomen!

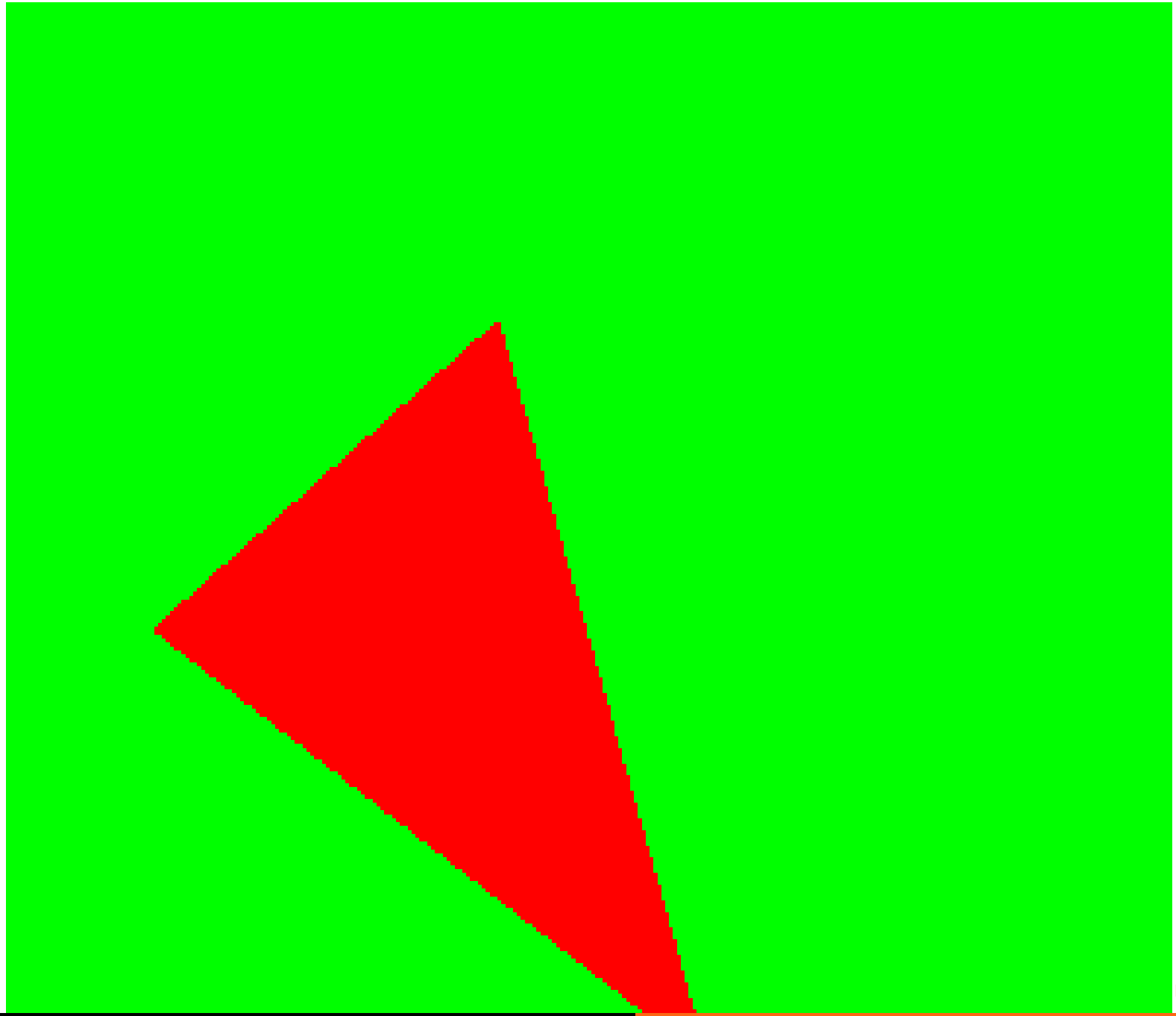


Inhalt der pdf-Datei:

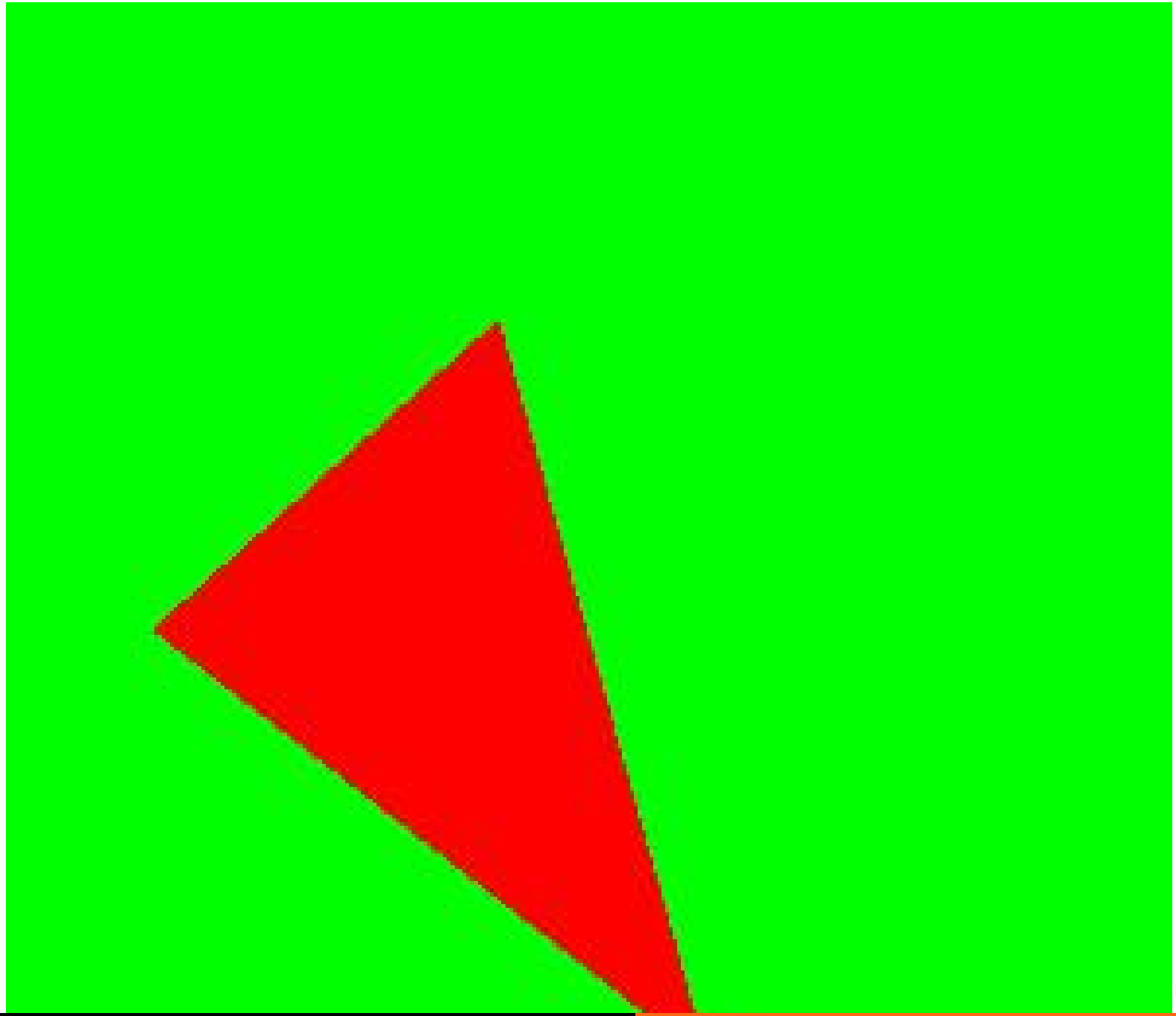
```
%PDF-1.4
3 0 obj
<< /Length 4 0 R
/Filter /FlateDecode
>>
stream
dfjkhgsdfkhsdfjkjhgksdfjh
endstream
endobj
4 0 obj
101
endobj
2 0 obj
<<
/ExtGState <<
/a0 << /CA 1 /ca 1 >>
>>
>>
endobj
5 0 obj
<< /Type /Page
/Parent 1 0 R
/MediaBox [ 0 0 267.428558 221.714279 ]
/Contents 3 0 R
/Group <<
/Type /Group
/S /Transparency
/CS /DeviceRGB
>>
/Resources 2 0 R
>>
endobj
1 0 obj
<< /Type /Pages
/Kids [ 5 0 R ]
/Count 1
>>
endobj
6 0 obj
<< /Creator (cairo 1.10.2 (http://cairographics.org))
/Producer (cairo 1.10.2 (http://cairographics.org))
>>
endobj
7 0 obj
<< /Type /Catalog
/Pages 1 0 R
>>
endobj
xref
0 8
0000000000 65535 f
0000000501 00000 n
0000000215 00000 n
0000000015 00000 n
0000000193 00000 n
0000000287 00000 n
0000000566 00000 n
0000000693 00000 n
trailer
<< /Size 8
/Root 7 0 R
/Info 6 0 R
>>
startxref
745

%%EOF
```

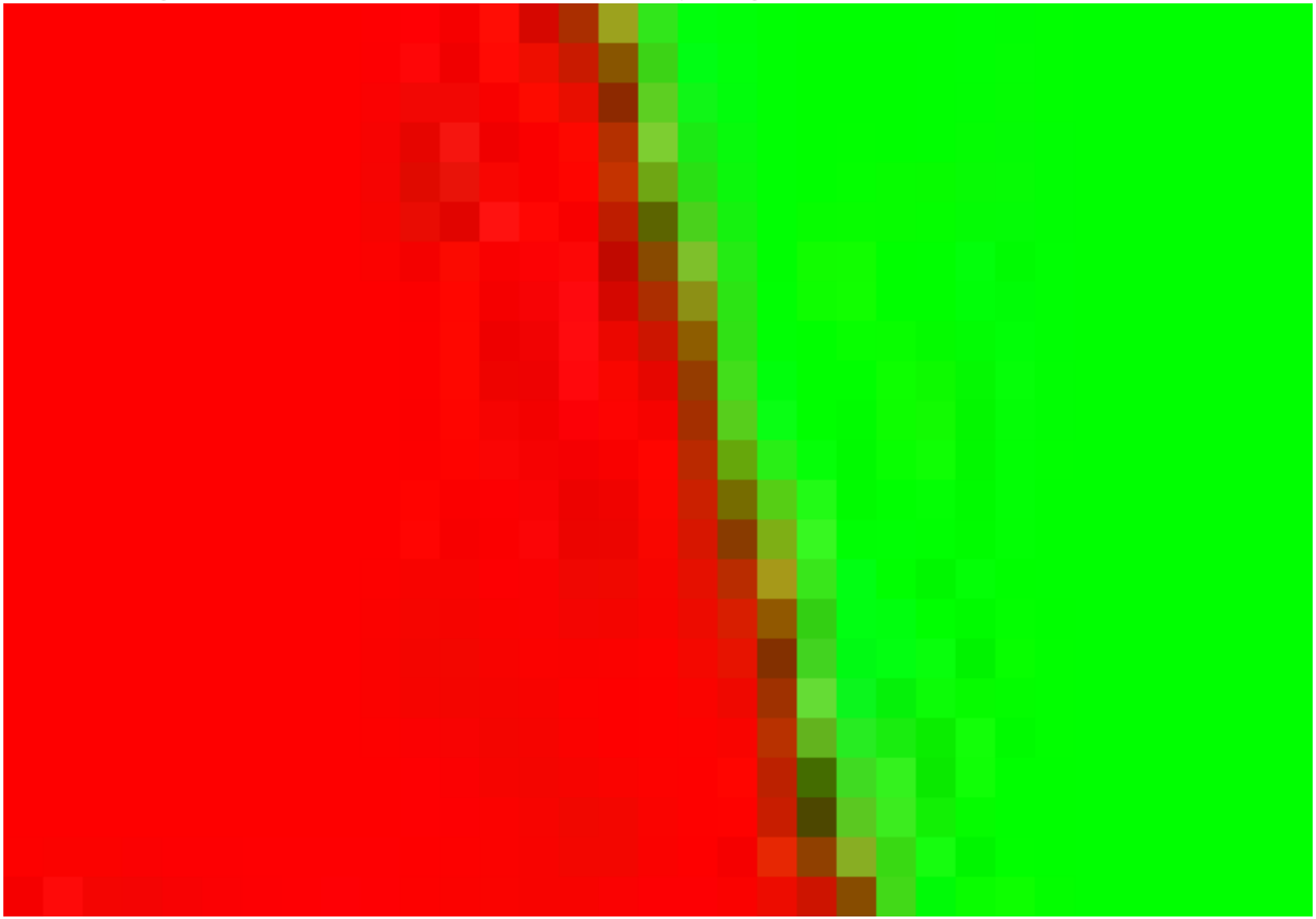
Dasselbe Bild als png gespeichert:



Dasselbe Bild als jpeg gespeichert:



Ein vergrößerter Ausschnitt des jpeg-Bilds:



PNG:

Zwei Phasen beim Speichern eines Bildes:

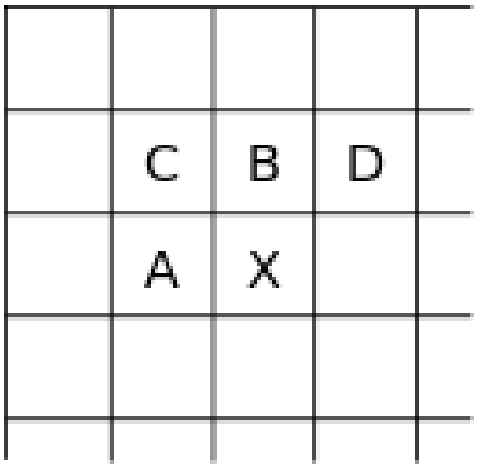
- Filtern (Hier: rate nächsten Farbwert, speichere Differenz)
- Komprimieren (erst LZ77, dann Huffman-Coding)

PNG:

Zwei Phasen beim Speichern eines Bildes:

- Filtern (Hier: rate nächsten Farbwert, speichere Differenz)
- Komprimieren (erst LZ77, dann Huffman-Coding)

1. Filtern: Durchlaufe das Bild zeilenweise Pixel für Pixel.
Rate die Farbe des aktuellen Pixels X aus den Nachbarn.



Vier Varianten:
An Stelle X: (A,B,C bereits bekannt)

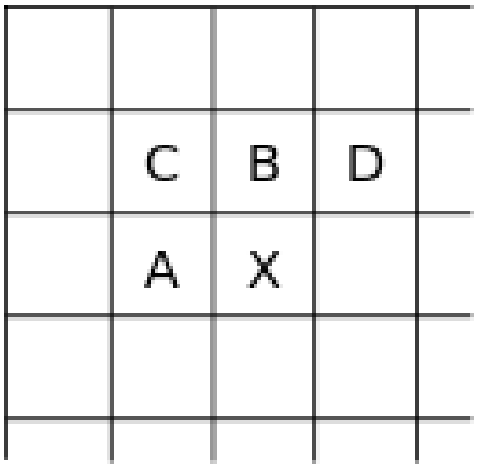
- Rate: A,
- Rate: B,
- Rate: $\frac{1}{2}$(A+B),

PNG:

Zwei Phasen beim Speichern eines Bildes:

- Filtern (Hier: rate nächsten Farbwert, speichere Differenz)
- Komprimieren (erst LZ77, dann Huffman-Coding)

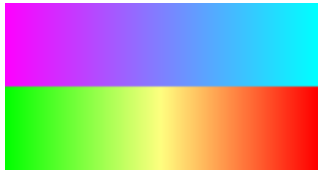
1. Filtern: Durchlaufe das Bild zeilenweise Pixel für Pixel.
Rate die Farbe des aktuellen Pixels X aus den Nachbarn.



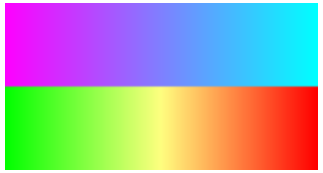
Vier Varianten:
An Stelle X: (A,B,C bereits bekannt)

- Rate: A,
- Rate: B,
- Rate: $\frac{1}{2}$(A+B),
- Rate: Wert, der der am nächsten an p =A+B-C ist.

Speichere die Differenz zwischen geratenem und korrektem Wert.

Warum ist das klug? In Bildern oft  oder 

Änderung (Differenz) von links nach rechts z.B.: 111111111...
(Farb*verläufe* zeigen evtl. mehr Regelmäßigkeit als Farbwerte)

Warum ist das klug? In Bildern oft  oder 

Änderung (Differenz) von links nach rechts z.B.: 111111111...
(Farb*verläufe* zeigen evtl. mehr Regelmäßigkeit als Farbwerte)

2. Komprimieren. Zunächst **LZ77**-Algorithmus:

Prinzip: Lesen wir einen String zum zweiten Mal, dann wird nur der Verweis gespeichert: Siehe Wort Nummer 44.

Genauer (dies ist nur ein einfach zu erklärendes Schema, das wahre Verfahren ist ein wenig komplizierter):

1. Lese String von links nach rechts, Zeichen für Zeichen.
2. Solange das aktuelle Wort bereits bekannt ist: weiterlesen.
3. Ist das aktuelle Wort unbekannt, wird es in einem **Wörterbuch** gespeichert, in der Form (Nummer, Nummer-bekanntes-Wort letzter-Buchstabe). Aktuelles Wort := $\varnothing$, weiterlesen (weiter bei 1.)

Bsp.: Komprimiere AABABBBABAABABBBABBABB.

Erstes Wort: A. Unbekannt, also: Wort 1 = A.

$$A|ABABBBABAABABBBABBABB$$

Bsp.: Komprimiere AABABBBABAABABBBABBABB.

Erstes Wort: A. Unbekannt, also: Wort 1 = A.

$$A|ABABBBABAABABBBABBABB$$

Zweites Wort: A. Bekannt, nichts tun.

$$A|ABABBBABAABABBBABBABB$$

Bsp.: Komprimiere AABABBBABAABABBBABBABB.

Erstes Wort: A. Unbekannt, also: Wort 1 = A.

$$A|ABABBBABAABABBBABBABB$$

Zweites Wort: A. Bekannt, nichts tun.

$$A|ABABBBABAABABBBABBABB$$

Drittes Wort: AB. Unbekannt, also: Wort 2 = 1B (= Wort 1-B).

$$A|AB|ABBBABAABABBBABBABB$$

Bsp.: Komprimiere AABABBBABAABABBBABBABB.

Erstes Wort: A. Unbekannt, also: Wort 1 = A.

$$A|ABABBBABAABABBBABBABB$$

Zweites Wort: A. Bekannt, nichts tun.

$$A|ABABBBABAABABBBABBABB$$

Drittes Wort: AB. Unbekannt, also: Wort 2 = 1B (= Wort 1-B).

$$A|AB|ABBBABAABABBBABBABB$$

Viertes Wort: A. Bekannt.
Fünftes Wort: AB, Bekannt.
Sechstes Wort ABB, unbekannt: also Wort 3 = 2B.

$$A|AB|ABB|BABAABABBBABBABB$$

Siebtes Wort: B, unbekannt: Wort 4 = ∅B (=Wort 0-B).

$$A|AB|ABB|B|ABAABABBBABBABB$$

Siebtes Wort: B, unbekannt: Wort 4 = ∅B (=Wort 0-B).

$$A|AB|ABB|B|ABAABABBBABBABB$$

Achtes Wort: A. Neuntes Wort: AB.
Zehntes Wort: ABA, unbekannt: Wort 5 = 2A.

$$A|AB|ABB|B|ABA|ABABBBABBABB$$

Siebtes Wort: B, unbekannt: Wort 4 = ∅B (=Wort 0-B).

$$A|AB|ABB|B|ABAABABBBABBABB$$

Achtes Wort: A. Neuntes Wort: AB.
Zehntes Wort: ABA, unbekannt: Wort 5 = 2A.

$$A|AB|ABB|B|ABA|ABABBBABBABB$$

Usw.

$$A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB$$

(Das letzte Wort BB ist bekannt, speichern wir trotzdem.) Also:

1	2	3	4	5	6	7	8	9
A	AB	ABB	B	ABA	$ABAB$	BB	$ABBA$	BB
$\varnothing A$	$1B$	$2B$	$\varnothing B$	$2A$	$5B$	$4B$	$3A$	$7-$

Siebtes Wort: B, unbekannt: Wort 4 = ∅B (=Wort 0-B).

$$A|AB|ABB|B|ABAABABBBABBABB$$

Achtes Wort: A. Neuntes Wort: AB.
Zehntes Wort: ABA, unbekannt: Wort 5 = 2A.

$$A|AB|ABB|B|ABA|ABABBBABBABB$$

Usw.

$$A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB$$

(Das letzte Wort BB ist bekannt, speichern wir trotzdem.) Also:

1	2	3	4	5	6	7	8	9
A	AB	ABB	B	ABA	$ABAB$	BB	$ABBA$	BB
$\varnothing A$	$1B$	$2B$	$\varnothing B$	$2A$	$5B$	$4B$	$3A$	$7-$

Wir speichern: $\varnothing A$ als $(000, 0)$, $1B$ als $(001, 1)$, $2B$ als $(010, 1)$, $\varnothing B$ als $(000, 1)$, $2A$ als $(010, 0)$ usw. Also:

$$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$$

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$$

Wort 1: A, also $A|$.

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$$

Wort 1: A, also $A|$.
Wort 2: $1B$, also AB, also $A|AB|$. (nicht AB!)

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$$

Wort 1: A, also $A|$.
Wort 2: $1B$, also AB, also $A|AB|$. (nicht AB!)
Wort 3: $2B$, also ABB, also $A|AB|ABB$.

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$$

Wort 1: A, also $A|$.
Wort 2: $1B$, also AB, also $A|AB|$. (nicht AB!)
Wort 3: $2B$, also ABB, also $A|AB|ABB$.
Wort 4: $\varnothing B$, also B, also $A|AB|ABB|B$.

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$$

Wort 1: A, also $A|$.
Wort 2: $1B$, also AB, also $A|AB|$. (nicht AB!)
Wort 3: $2B$, also ABB, also $A|AB|ABB$.
Wort 4: $\varnothing B$, also B, also $A|AB|ABB|B$.
Wort 5: $2A$, also ABA, also $A|AB|ABB|B|ABA$.

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$$

Wort 1: A, also $A|$.
Wort 2: $1B$, also AB, also $A|AB|$. (nicht AB!)
Wort 3: $2B$, also ABB, also $A|AB|ABB$.
Wort 4: $\varnothing B$, also B, also $A|AB|ABB|B$.
Wort 5: $2A$, also ABA, also $A|AB|ABB|B|ABA$.
Wort 6: $5B$, also $ABAB$, also $A|AB|ABB|B|ABA|ABAB$.

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$$

Wort 1: A, also $A|$.
Wort 2: $1B$, also AB, also $A|AB|$. (nicht AB!)
Wort 3: $2B$, also ABB, also $A|AB|ABB$.
Wort 4: $\varnothing B$, also B, also $A|AB|ABB|B$.
Wort 5: $2A$, also ABA, also $A|AB|ABB|B|ABA$.
Wort 6: $5B$, also $ABAB$, also $A|AB|ABB|B|ABA|ABAB$.
Wort 7: $4B$, also BB, also $A|AB|ABB|B|ABA|ABAB|BB$.

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$$

Wort 1: A, also $A|$.
Wort 2: $1B$, also AB, also $A|AB|$. (nicht AB!)
Wort 3: $2B$, also ABB, also $A|AB|ABB$.
Wort 4: $\varnothing B$, also B, also $A|AB|ABB|B$.
Wort 5: $2A$, also ABA, also $A|AB|ABB|B|ABA$.
Wort 6: $5B$, also $ABAB$, also $A|AB|ABB|B|ABA|ABAB$.
Wort 7: $4B$, also BB, also $A|AB|ABB|B|ABA|ABAB|BB$.
Wort 8: $3A$, also $ABBA$, also $A|AB|ABB|B|ABA|ABAB|BB|ABBA$.

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$

Wort 1: A, also $A|$.
Wort 2: $1B$, also AB, also $A|AB|$. (nicht AB!)
Wort 3: $2B$, also ABB, also $A|AB|ABB$.
Wort 4: $\varnothing B$, also B, also $A|AB|ABB|B$.
Wort 5: $2A$, also ABA, also $A|AB|ABB|B|ABA$.
Wort 6: $5B$, also $ABAB$, also $A|AB|ABB|B|ABA|ABAB$.
Wort 7: $4B$, also BB, also $A|AB|ABB|B|ABA|ABAB|BB$.
Wort 8: $3A$, also $ABBA$, also $A|AB|ABB|B|ABA|ABAB|BB|ABBA$.
Wort 9: $7-$, also $BB-$, also $A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB$.

Oben nur Buchstaben A,B, also nur ein bit pro Buchstabe. Mehr Buchstaben, mehr bits.

Dekodieren:

$(000, 0), (001, 1), (010, 1), (000, 1), (010, 0), (101, 1), (100, 1), (011, 0), (111, -)$

Wort 1: A, also $A|$.
Wort 2: $1B$, also AB, also $A|AB|$. (nicht AB!)
Wort 3: $2B$, also ABB, also $A|AB|ABB$.
Wort 4: $\varnothing B$, also B, also $A|AB|ABB|B$.
Wort 5: $2A$, also ABA, also $A|AB|ABB|B|ABA$.
Wort 6: $5B$, also $ABAB$, also $A|AB|ABB|B|ABA|ABAB$.
Wort 7: $4B$, also BB, also $A|AB|ABB|B|ABA|ABAB|BB$.
Wort 8: $3A$, also $ABBA$, also $A|AB|ABB|B|ABA|ABAB|BB|ABBA$.
Wort 9: $7-$, also $BB-$, also $A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB$.

Das hier zeigt die Idee, ist einfach zu erklären und zu analysieren.

Wenn man dies verstanden hat, ist die reale Variante schnell zu verstehen.

LZ77 jetzt echt:

Speichere statt der Wörterbuchnummer gesehene Wörter als: Position, Länge.
"Position n" heißt: gehe n Zeichen zurück.

Merke Dir aber alle bereits gesehenen Wörter (nicht nur die im Wörterbuch).

Hier am Beispiel erklärt (Pseudocode u.a. siehe wikipedia)

Komprimiere AACAACABCABAAAC.

LZ77 jetzt echt:

Speichere statt der Wörterbuchnummer gesehene Wörter als: Position, Länge.
"Position n" heißt: gehe n Zeichen zurück.

Merke Dir aber alle bereits gesehenen Wörter (nicht nur die im Wörterbuch).

Hier am Beispiel erklärt (Pseudocode u.a. siehe wikipedia)

Komprimiere AACAACABCABAAAC.

Erstes Wort: A. Kam noch nicht vor, also speichern: (0,0,A).

$$A|\underline{A}CAACABCABAAAC$$

Zweites Wort A, kam schon vor, nichts tun.

$$A|A\underline{C}AACABCABAAAC$$

Drittes Wort AC, kam noch nicht vor, also speichern: (1,1,C):

LZ77 jetzt echt:

Speichere statt der Wörterbuchnummer gesehene Wörter als: Position, Länge.
"Position n" heißt: gehe n Zeichen zurück.

Merke Dir aber alle bereits gesehenen Wörter (nicht nur die im Wörterbuch).

Hier am Beispiel erklärt (Pseudocode u.a. siehe wikipedia)

Komprimiere AACAACABCABAAAC.

Erstes Wort: A. Kam noch nicht vor, also speichern: (0,0,A).

$$A|\underline{A}CAACABCABAAAC$$

Zweites Wort A, kam schon vor, nichts tun.

$$A|A\underline{C}AACABCABAAAC$$

Drittes Wort AC, kam noch nicht vor, also speichern: (1,1,C): Das Wort an Position 1 der Länge 1 ist A, also ist "AC" gespeichert.

$$A|AC|\underline{A}ACABCABAAAC$$

Viertes Wort A, kam schon vor, nichts tun.

$$A|AC|A\underline{A}CABCABAAAC$$

Fünftes Wort AA, kam schon vor (obwohl nicht im Wörterbuch!).

$$A|AC|\underline{A}ACABCABAAAC$$

Viertes Wort A, kam schon vor, nichts tun.

$$A|AC|A\underline{A}CABCABAAAC$$

Fünftes Wort AA, kam schon vor (obwohl nicht im Wörterbuch!).

$$A|AC|AA\underline{C}ABCABAAAC$$

Sechstes Wort AAC, kam schon vor, nichts tun.

$$A|AC|AAC\underline{A}BCABAAAC$$

Siebtes Wort AACA, kam schon vor (!)

$$A|AC|\underline{A}ACABCABAAAC$$

Viertes Wort A, kam schon vor, nichts tun.

$$A|AC|A\underline{A}CABCABAAAC$$

Fünftes Wort AA, kam schon vor (obwohl nicht im Wörterbuch!).

$$A|AC|AA\underline{C}ABCABAAAC$$

Sechstes Wort AAC, kam schon vor, nichts tun.

$$A|AC|AAC\underline{A}BCABAAAC$$

Siebtes Wort AACA, kam schon vor (!)

$$A|AC|AACA\underline{B}CABAAAC$$

Achtes Wort AACAB, neu, also speichern: (3,4,B). (! unser AACA ist um 3 gegen das erste AACA) verschoben!)

$$A|AC|AACAB|\underline{C}ABAAAC$$

C, CA, CAB kamen schon vor. CABA ist neu, also speichern: (3,3,A).

$$A|AC|AACAB|CABA|\underline{A}AC$$

A, AA kamen schon vor. AAC auch schon, ist aber letztes Wort, also speichern: (12,3,/).

$$A|AC|AACAB|\underline{C}ABAAAC$$

C, CA, CAB kamen schon vor. CABA ist neu, also speichern: (3,3,A).

$$A|AC|AACAB|CABA|\underline{A}AC$$

A, AA kamen schon vor. AAC auch schon, ist aber letztes Wort, also speichern: (12,3,/).

Rückübersetzen:

(0,0,A) (1,1,C) (3,4,B) (3,3,A) (12,3,/):

A

$$A|AC|AACAB|\underline{C}ABAAAC$$

C, CA, CAB kamen schon vor. CABA ist neu, also speichern: (3,3,A).

$$A|AC|AACAB|CABA|\underline{A}AC$$

A, AA kamen schon vor. AAC auch schon, ist aber letztes Wort, also speichern: (12,3,/).

Rückübersetzen:

(0,0,A) (1,1,C) (3,4,B) (3,3,A) (12,3,/):

A AC

$$A|AC|AACAB|\underline{C}ABAAAC$$

C, CA, CAB kamen schon vor. CABA ist neu, also speichern: (3,3,A).

$$A|AC|AACAB|CABA|\underline{A}AC$$

A, AA kamen schon vor. AAC auch schon, ist aber letztes Wort, also speichern: (12,3,/).

Rückübersetzen:

(0,0,A) (1,1,C) (3,4,B) (3,3,A) (12,3,/):

A AC AACAB

$$A|AC|AACAB|\underline{C}ABAAAC$$

C, CA, CAB kamen schon vor. CABA ist neu, also speichern: (3,3,A).

$$A|AC|AACAB|CABA|\underline{A}AC$$

A, AA kamen schon vor. AAC auch schon, ist aber letztes Wort, also speichern: (12,3,/).

Rückübersetzen:

(0,0,A) (1,1,C) (3,4,B) (3,3,A) (12,3,/):

A AC AACAB CABA

$$A|AC|AACAB|\underline{C}ABAAAC$$

C, CA, CAB kamen schon vor. CABA ist neu, also speichern: (3,3,A).

$$A|AC|AACAB|CABA|\underline{A}AC$$

A, AA kamen schon vor. AAC auch schon, ist aber letztes Wort, also speichern: (12,3,/).

Rückübersetzen:

(0,0,A) (1,1,C) (3,4,B) (3,3,A) (12,3,/):

A AC AACAB CABA AAC

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A B

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A B A

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A B AB

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A B ABA

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A B ABAB

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A B ABABA

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A B ABABAB

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A B ABABABA

Auf diese Weise ist RLE eingebaut! Denn:

Komprimiere ABABABABAB.

Erstes Wort A, also (0,0,A)

$$A|BABABABAB$$

Zweites Wort B, also (0,0,B)

$$AB|ABABABAB$$

A, AB, ABA, ABAB... kam schon vor! Also: (2,8,/)

Rückübersetzen: (0,0,A) (0,0,B) (2,8,/):

A B ABABABAB

...zurück zum Komprimieren bei png. Nächster Schritt:

Huffman coding: Speichere häufige Symbole mit wenig Bits.
Beispiel: Nachricht: 11112233455555666666

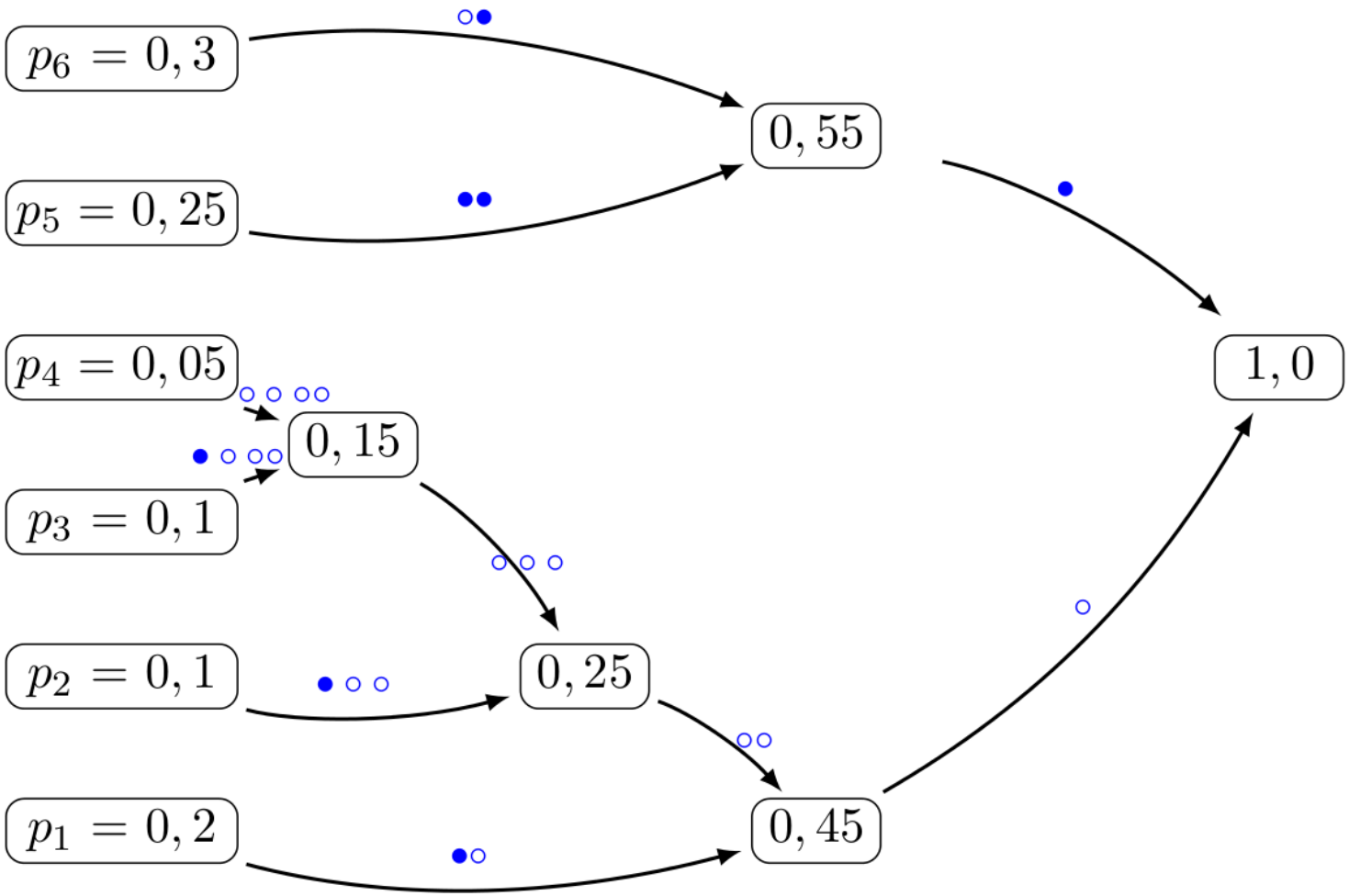
Naiv: 20 Zeichen. 8 Bits pro Zeichen macht 160 Bits.

Normal: Nur 6 verschiedene Zeichen, also nur 3 Bits pro Zeichen nötig. Also $20 \cdot 3 = 60$ Bit Speicherbedarf.

Clever: Huffman-codiert:

●○|●○|●○|●○|●○○|●○○|●○○○|●○○○|○○○○|●●|●●|●●|●●|●●|○●|○●|○●|○●|○●|○●|

$4 \cdot 2 + 2 \cdot 3 + 2 \cdot 4 + 1 \cdot 4 + 5 \cdot 2 + 6 \cdot 2 = 48$ Bit Speicher.
Im Schnitt 2,4 Bit pro Wort.


$p_6 = 0,3$
$p_5 = 0,25$
$p_4 = 0,05$
$p_3 = 0,1$
$p_2 = 0,1$
$p_1 = 0,2$
0, 55
0, 15
0, 25
0, 45
1, 0

png vs jpg vs gif vs ...

png speichert verlustfrei (**lossless**, keine Information geht verloren).

Es kann auch stärker komprimieren, wenn man verlustbehaftetes (**lossy**) Komprimieren vorwählt: Dann wird die Zahl der Farben reduziert.

Die Leistung (Komprimierungsgrad) lässt sich daher schwierig vergleichen. **jpeg** ist lossy, aber klein, insbesondere bei Fotos. **png** ist verlustfrei und klein bei wenigen Farben (s.o.), aber groß bei vielen Farben (Fotos!)

png vs jpg vs gif vs ...

png speichert verlustfrei (**lossless**, keine Information geht verloren).

Es kann auch stärker komprimieren, wenn man verlustbehaftetes (**lossy**) Komprimieren vorwählt: Dann wird die Zahl der Farben reduziert.

Die Leistung (Komprimierungsgrad) lässt sich daher schwierig vergleichen. **jpeg** ist lossy, aber klein, insbesondere bei Fotos.
png ist verlustfrei und klein bei wenigen Farben (s.o.), aber groß bei vielen Farben (Fotos!)

png to gif: png kann 2^{24} Farben, gif nur 2^8. Daher wird beim Konvertieren die Datei evtl. kleiner, aber lossy!
Dafür kann gif Animationen.

Andere Formate wie **tiff** enthalten viel mehr Informationen (sehr viele Farben, mehrere Ebenen ("layer"), Transparenz, cmyk, Metadaten...), sind dafür größer.

cmyk: Cyan-Maganta-Yellow-blacK: braucht der Drucker.
Schwarz in RGB: 000000 (keine Farbe, also Bildschirm schwarz)
Aber: keine Farbe auf dem Drucker = weiß.

Nebenbei... wofür stehen die Abkürzungen?

- pdf:

cmyk: Cyan-Maganta-Yellow-blacK: braucht der Drucker.
Schwarz in RGB: 000000 (keine Farbe, also Bildschirm schwarz)
Aber: keine Farbe auf dem Drucker = weiß.

Nebenbei... wofür stehen die Abkürzungen?

- pdf: *portable document format*
- png:

cmyk: Cyan-Maganta-Yellow-blacK: braucht der Drucker.
Schwarz in RGB: 000000 (keine Farbe, also Bildschirm schwarz)
Aber: keine Farbe auf dem Drucker = weiß.

Nebenbei... wofür stehen die Abkürzungen?

- pdf: *portable document format*
- png: *portable network graphics*
- gif:

cmyk: Cyan-Maganta-Yellow-blacK: braucht der Drucker.
Schwarz in RGB: 000000 (keine Farbe, also Bildschirm schwarz)
Aber: keine Farbe auf dem Drucker = weiß.

Nebenbei... wofür stehen die Abkürzungen?

- pdf: *portable document format*
- png: *portable network graphics*
- gif: *graphics interchange format*
- jpg:

cmyk: Cyan-Maganta-Yellow-blacK: braucht der Drucker.
Schwarz in RGB: 000000 (keine Farbe, also Bildschirm schwarz)
Aber: keine Farbe auf dem Drucker = weiß.

Nebenbei... wofür stehen die Abkürzungen?

- pdf: *portable document format*
- png: *portable network graphics*
- gif: *graphics interchange format*
- jpg: *joint photographic experts group*