Компьютерная графика

Алгоритмы сжатия изображений

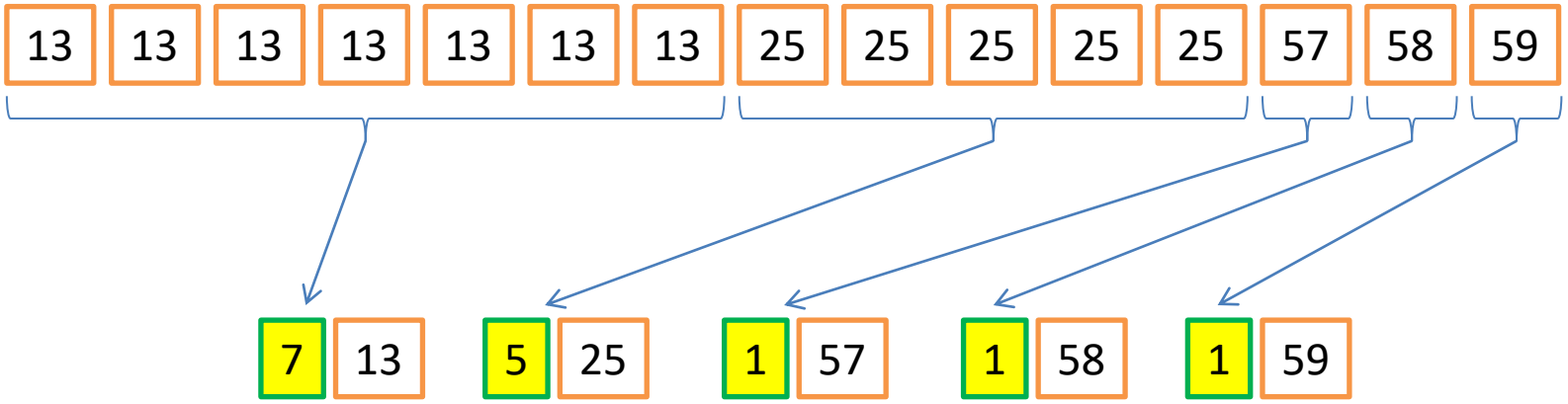
Сжатие без потерь

Алгоритм группового кодирования RLE (Run Length Encoding = кодирование длинами серий):

- Самый простой алгоритмом сжатия.
- Очень быстрый, понятный и в некоторых случаях достаточно эффективный.
- Используется как часть более сложных алгоритмов.

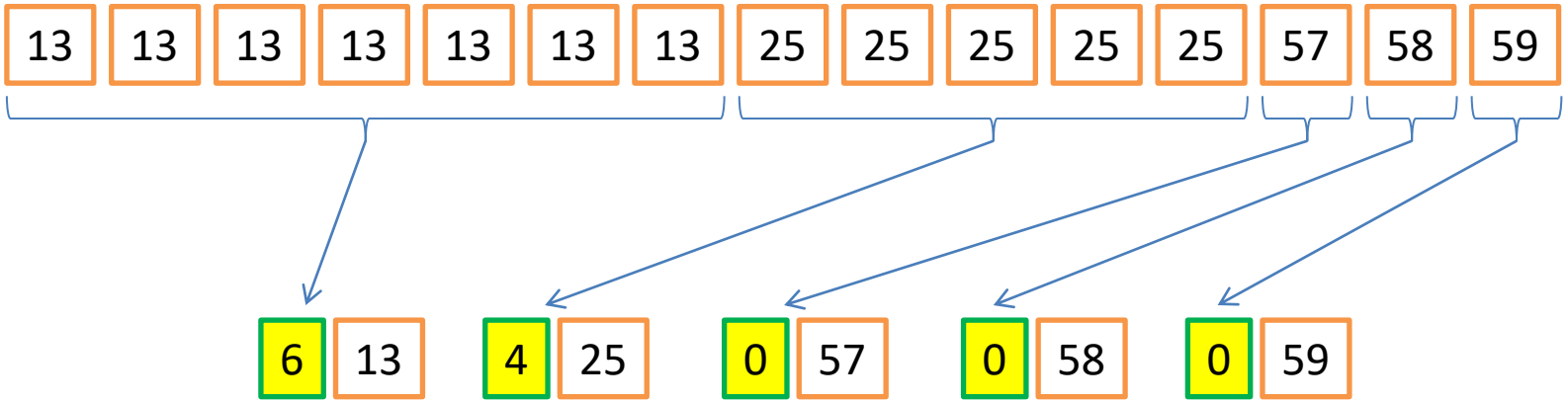
RLE (Run Length Encoding)

Последовательность одинаковых данных заменяется счетчиком повторений и одним экземпляром данных.



RLE (Run Length Encoding)

Последовательность одинаковых данных заменяется счетчиком повторений и одним экземпляром данных.



Нет смысла передавать нулевое значение счетчика !!!

RLE (Run Length Encoding)

Коэффициент сжатия RLE

- n = разрядность счетчика
- m = разрядность данных
- Максимальный коэффициент сжатия

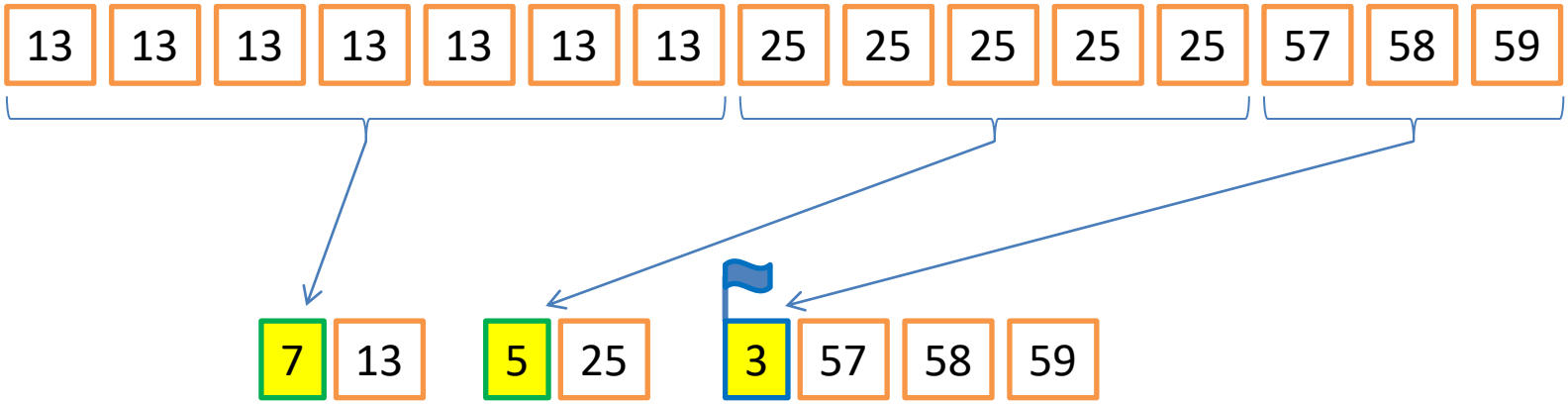
$$k_{max} = \frac{m \cdot 2^n}{n + m} \qquad k_{max} = \frac{8 \cdot 2^8}{8 + 8} = \frac{2048}{16} = 128$$

- Минимальный коэффициент сжатия

$$k_{min} = \frac{m}{n + m} \qquad k_{min} = \frac{8}{8 + 8} = 0.5$$ ❗❗❗

RLE с флагом

Старший бит счетчика повторений НЕ равен нулю (стоит флаг), если далее идут разные данные и равен нулю (флага нет), если данные одинаковые:



RLE с флагом

Коэффициент сжатия RLE с флагом

- n = разрядность счетчика
- m = разрядность данных
- Максимальный коэффициент сжатия

$$k_{max} = \frac{m \cdot 2^n}{n + m} \qquad k_{max} = \frac{8 \cdot 2^7}{8 + 8} = \frac{1024}{16} = 64$$

- Минимальный коэффициент сжатия

$$k_{min} = \frac{m \cdot 2^{n-1}}{n + m \cdot 2^{n-1}} \qquad k_{min} = \frac{8 \cdot 2^7}{8 + 8 \cdot 2^7} = \frac{1024}{1032} = 0.99$$

Кодирование изображений

Факс – передача черно-белых изображений

- Разрядность данных – 1 бит
- Значение данных меняется на каждом счетчике (имеет смысл передавать нулевое значение счетчика), начиная с белого.
- Сжатие по вертикали – следующая строка передает отличия от предыдущей (начиная с белой строки).

Кодирование RLE

- Можно ли сжимать изображение алгоритмом RLE с потерей данных?

LZW

- Авторы: Lempel - Ziv - Welch
- Используется в архиваторах (ZIP, ARJ, RAR)
- Сжатие потока данных в виде «цепочек»
- Строится динамическая таблица «цепочек»

Дано:

- Алфавит – A, B, C
- Входной поток «букв» - A,B,A,B,C,A,B,C,B…

LZW

Ключевым элементом алгоритма является таблица цепочек

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4		
5		
6		
7		
8		

Цепочка кодируется последним символом и ссылкой на начало цепочки, записанной ранее в таблицу, поэтому таблица содержит две колонки данных:

1) индекс начала цепочки;
2) символ, завершающий цепочку.

Нулевой индекс используется для завершения цепочки – далее у цепочки начала нет.

В самом начале таблица заполняется нулевой цепочкой (индекс ячейки 0) и единичными цепочками (по одной букве алфавита – ячейки 1..3)

Кодирование LZW (очередной шаг)

Кодер формирует цепочку из индекса найденной ранее цепочки (изначально равен нулю) и текущего входного символа, ищет ее в таблице и если находит, то запоминает ее индекс, иначе добавляет новую цепочку в таблицу и передает ее на выход

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4		
5		
6		
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу

Кодирование LZW (шаг 1)

- Вход: A
- Выход:

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4		
5		
6		
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу
"A"	0A	1		

Кодирование LZW (шаг 2)

- Вход: A,B
- Выход: 1,B

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5		
6		
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу
"A"	0A	1		
"AB"	1B	нет	"1B"	[#4] = 1 & B

Кодирование LZW (шаг 3)

- Вход: A,B,A
- Выход: 1,B

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5		
6		
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу
"A"	0A	1		
"AB"	1B	нет	"1B"	[#4] = 1 & B
"A"	0A	1		

Кодирование LZW (шаг 4)

- Вход: A,B,A,B
- Выход: 1,B

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5		
6		
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу
"A"	0A	1		
"AB"	1B	нет	"1B"	[#4] = 1 & B
"A"	0A	1		
"AB"	1B	4		

Кодирование LZW (шаг 5)

- Вход: A,B,A,B,C
- Выход: 1,B,4,C

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5	4	C
6		
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу
"A"	0A	1		
"AB"	1B	нет	"1B"	[#4] = 1 & B
"A"	0A	1		
"AB"	1B	4		
"ABC"	4C	нет	"4C"	[#5] = 4 & C

Кодирование LZW (шаг 6)

- Вход: A,B,A,B,C,A
- Выход: 1,B,4,C

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5	4	C
6		
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу
"A"	0A	1		
"AB"	1B	нет	"1B"	[#4] = 1 & B
"A"	0A	1		
"AB"	1B	4		
"ABC"	4C	нет	"4C"	[#5] = 4 & C
"A"	0A	1		

Кодирование LZW (шаг 7)

- Вход: A,B,A,B,C,A,B
- Выход: 1,B,4,C

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5	4	C
6		
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу
"A"	0A	1		
"AB"	1B	нет	"1B"	[#4] = 1 & B
"A"	0A	1		
"AB"	1B	4		
"ABC"	4C	нет	"4C"	[#5] = 4 & C
"A"	0A	1		
"AB"	1B	4		

Кодирование LZW (шаг 8)

- Вход: A,B,A,B,C,A,B,C
- Выход: 1,B,4,C

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5	4	C
6		
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу
"A"	0A	1		
"AB"	1B	нет	"1B"	[#4] = 1 & B
"A"	0A	1		
"AB"	1B	4		
"ABC"	4C	нет	"4C"	[#5] = 4 & C
"A"	0A	1		
"AB"	1B	4		
"ABC"	4C	5		

Кодирование LZW (шаг 9)

- Вход: A,B,A,B,C,A,B,C,B
- Выход: 1,B,4,C,5,B

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5	4	C
6	5	B
7		
8		

Входные данные	Текущая цепочка	Индекс текущей цепочки	Выход	Добавление в таблицу
"A"	0A	1		
"AB"	1B	нет	"1B"	[#4] = 1 & B
"A"	0A	1		
"AB"	1B	4		
"ABC"	4C	нет	"4C"	[#5] = 4 & C
"A"	0A	1		
"AB"	1B	4		
"ABC"	4C	5		
"ABCB"	5B	нет	"5B"	[#6] = 5 & B

Декодирование LZW

Декодер строит аналогичную таблицу цепочек и также инициализирует ее

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4		
5		
6		
7		
8		

1) Декодер считывает со входа пару - индекс и символ.
2) Добавляет их в таблицу (в очередную свободную ячейку).
3) Выдает на выход всю цепочку (разворачивая ее с конца в начало)

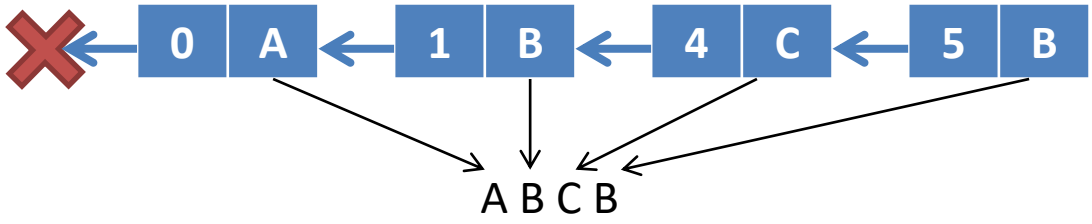
Декодирование LZW

Декодер строит аналогичную таблицу цепочек и также инициализирует ее

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5	4	C
6	5	B
7		
8		

1) Декодер считывает со входа пару - индекс и символ.
2) Добавляет их в таблицу (в очередную свободную ячейку).
3) Выдает на выход всю цепочку (разворачивая ее с конца в начало)

Например, цепочка 5B:



Декодирование LZW (шаг 1)

- Вход: 1,B
- Выход: A,B

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5		
6		
7		
8		

Входные данные	Добавление в таблицу	Выход
1B	[#4] = 1 & B	"AB"

Декодирование LZW (шаг 2)

- Вход: 1,B,4,C
- Выход: A,B,A,B,C

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5	4	C
6		
7		
8		

Входные данные	Добавление в таблицу	Выход
1B	[#4] = 1 & B	"AB"
4C	[#5] = 4 & C	"ABC"

Декодирование LZW (шаг 3)

- Вход: 1,B,4,C,5,B
- Выход: A,B,A,B,C,A,B,C,B

Индекс ячейки	Индекс начала цепочки	Символ в конце цепочки
0	0	**0**
1	0	**A**
2	0	**B**
3	0	**C**
4	1	B
5	4	C
6	5	B
7		
8		

Входные данные	Добавление в таблицу	Выход
1B	[#4] = 1 & B	"AB"
4C	[#5] = 4 & C	"ABC"
5B	[#6] = 5 & B	"ABCB"

Вопросы

1. Что делает кодер, когда заканчивается таблица (добавлена цепочка в последнюю свободную ячейку)?
2. Что делает декодер в той же ситуации?
3. Как декодер синхронизуется к кодеру?
4. Как связаны разрядность индекса в таблице с разрядностью алфавита?

Коэффициент сжатия LZW

- Пусть разрядность алфавита 8 бит, разрядность индекса 12 бит.

- Для 2-х символов $k_2 = \frac{2 * 8}{12 + 8} = 0.8$!!!

- Для 3-х символов $k_3 = \frac{3 * 8}{12 + 8} = 1.2$???

- Для 4-х символов $k_4 = \frac{4 * 8}{12 + 8} = 1.6$

Алгоритм Хаффмана

- Статистический алгоритм упаковки данных без потерь.
- Строит таблицу соответствия каждому символу алфавита уникальной последовательности бит разной длины.
- Для часто встречающихся символов длина последовательности меньше.

Алгоритм Хаффмана

Строится двоичное дерево, в котором

- каждый узел имеет суммарную вероятность всех дочерних узлов;
- каждый лист (конечный узел) содержит один символ.

Обход всего дерева дает таблицу Хаффмана

Алгоритм Хаффмана

- Сначала добавим в корневой узел листья со всеми символами алфавита.
- Затем итеративно повторяем:
  - найдем два узла с наименьшей вероятностью и вынесем их в отельный узел, у которого в качестве вероятности запишем сумму вероятностей найденных узлов
- пока в корневом узле не останется два узла.

Арифметическое кодирование

- Статистический алгоритм упаковки данных без потерь позволяет достичь максимальной степени сжатия при длине данных $-\log_2(S)$ бит (**дробное число бит**) для символа с вероятностью S.

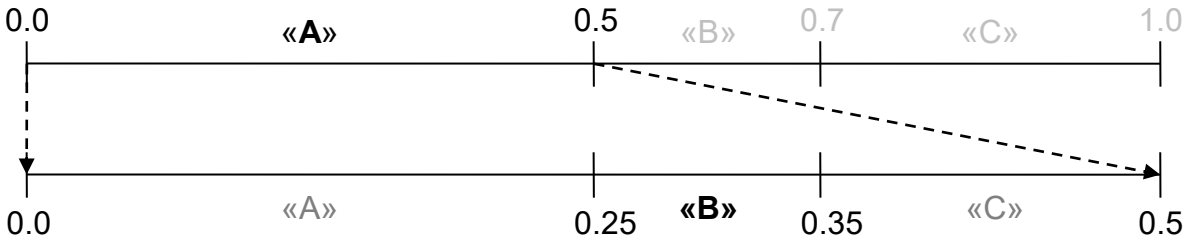
- Алгоритмы арифметического кодирования кодируют входной поток в очень длинное дробное число в интервале от 0.0 до 1.0

Арифметическое кодирование

- Рассмотрим пример кодирования последовательности «ABCABC», если вероятности символов следующие:
  A – 0.5, B – 0.2, C – 0.3.
- В арифметическом кодере каждый символ представляется интервалом в диапазоне чисел [0, 1) в соответствии с частотой его появления:

Арифметическое кодирование

- Первый входной символ «A» переводит начальный интервал [0, 1) в интервал [0, 0.5)



- Второй входной символ «B» переводит интервал [0, 0.5) в интервал [0.25, 0.35)
- и так далее интервал сужается и сужается

Арифметическое кодирование

Для «ABCABC» получаем интервал [0.3296, 0.3305)

№	Символ	L	R	Границы
1	**A**	0	0.5	0+(1-0)*0.0 = 0 0+(1-0)*0.5 = 0.5
2	**B**	0.5	0.7	0+(0.5-0)*0.5 = 0.25 0+(0.5-0)*0.7 = 0.35
3	**C**	0.7	1	0.25+(0.35-0.25)*0.7 = 0.32 0.25+(0.35-0.25)*1.0 = 0.35
4	**A**	0	0.5	0.32+(0.35-0.32)*0.0 = 0.32 0.32+(0.35-0.32)*0.5 = 0.335
5	**B**	0.5	0.7	0.32+(0.335-0.32)*0.5 = 0.3275 0.32+(0.335-0.32)*0.7 = 0.3305
6	**C**	0.7	1	0.3275+(0.3305-0.3275)*0.7 = **0.3296** 0.3275+(0.3305-0.3275)*1.0 = **0.3305**

ширина интервала 0,0009 – 10 бит достаточно для кодирования

Передаем число 0,330078 = 338/1024

Арифметическое кодирование

Декодер начинает с интервала [0, 1)

1. Число 0,330078 лежит в диапазоне [0, 0.5), это значит, что первым идет символ «A».
2. Далее интервал текущего символа растягивается до интервала [0, 1) и цикл повторяется.

№	Символ	L	R	
1	**A**	0	0,5	(**0,330078**-0)/(0,5-0) = 0,660156
2	**B**	0,5	0,7	(0,660156-0,5)/(0,7-0,5) = 0,800780
3	**C**	0,7	1	(0,800780-0,7)/(1-0,7) = 0,335933
4	**A**	0	0,5	(0,335933-0)/(0,5-0) = 0,671867
5	**B**	0,5	0,7	(0,671867-0,5)/(0,7-0,5) = 0,859333
6	**C**	0,7	1	(0,859333-0,7)/(1-0,7) = 0,531111

JPEG

- Сжатие с потерей данных !!!
- Характерное сжатие в 10 раз
- Базируется на ДКП (дискретное косинусное преобразование), являющееся «реальной частью» Фурье-преобразования (без фазы)
- Симметричный алгоритм – сложность кодирования и сложность декодирования примерно равны

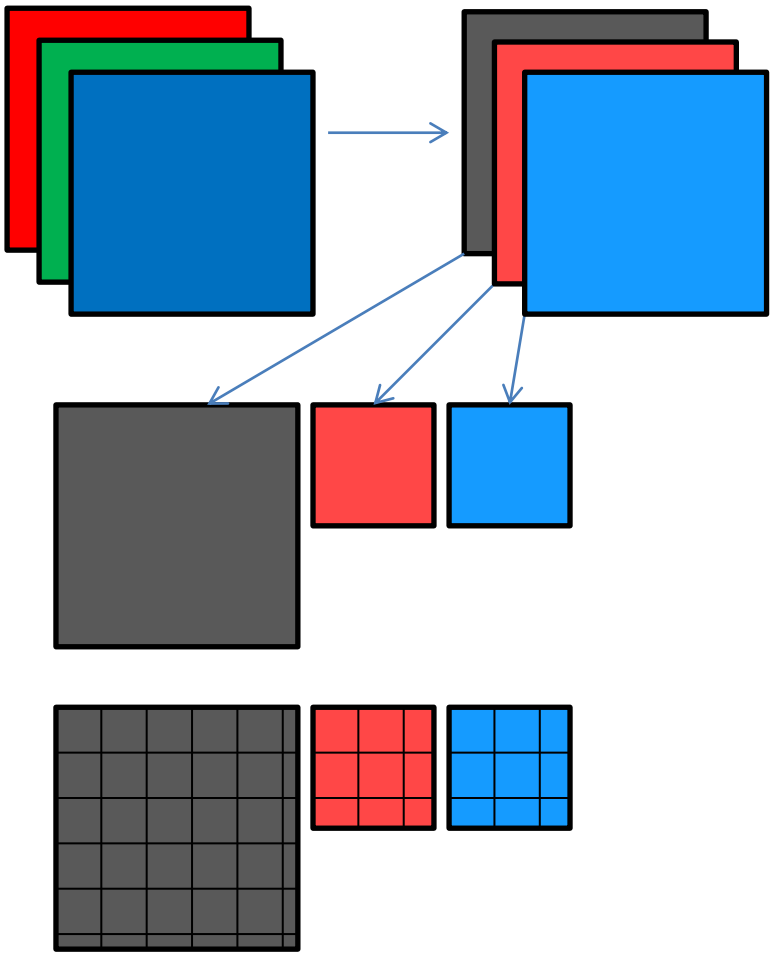
Кодирование JPEG

Шаг 1. Матрицирование (попиксельно)

Y = 0.299*R + 0.587*G + 0.114*B

Cr = R - Y

Cb = B - Y



Шаг 2. Децимация (прореживание) компонент Cr и Cb (в 4 раза)

Далее данные разбиваются на блоки 8x8 и обрабатываются независимо – сначала все Y, затем все Cr, затем все Cb

Кодирование JPEG

Шаг 3. Двумерное ДКП 8x8

(дискретное косинусное преобразование)

V00	V10	V20	V30	V40	V50	V60	V70
V01	V11	V21	V31	V41	V51	V61	V71
V02	V12	V22	V32	V42	V52	V62	V72
V03	V13	V23	V33	V43	V53	V63	V73
V04	V14	V24	V34	V44	V54	V64	V74
V05	V15	V25	V35	V45	V55	V65	V75
V06	V16	V26	V36	V46	V56	V66	V76
V07	V17	V27	V37	V47	V57	V67	V77

C00	C10	C20	C30	C40	C50	C60	C70
C01	C11	C21	C31	C41	C51	C61	C71
C02	C12	C22	C32	C42	C52	C62	C72
C03	C13	C23	C33	C43	C53	C63	C73
C04	C14	C24	C34	C44	C54	C64	C74
C05	C15	C25	C35	C45	C55	C65	C75
C06	C16	C26	C36	C46	C56	C66	C76
C07	C17	C27	C37	C47	C57	C67	C77

$$C(i,j) = \frac{A(i)A(j)}{\sqrt{2N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} V(x,y) \cdot \cos\left(\frac{\pi i(2x+1)}{2N}\right) \cos\left(\frac{\pi j(2y+1)}{2N}\right)$$

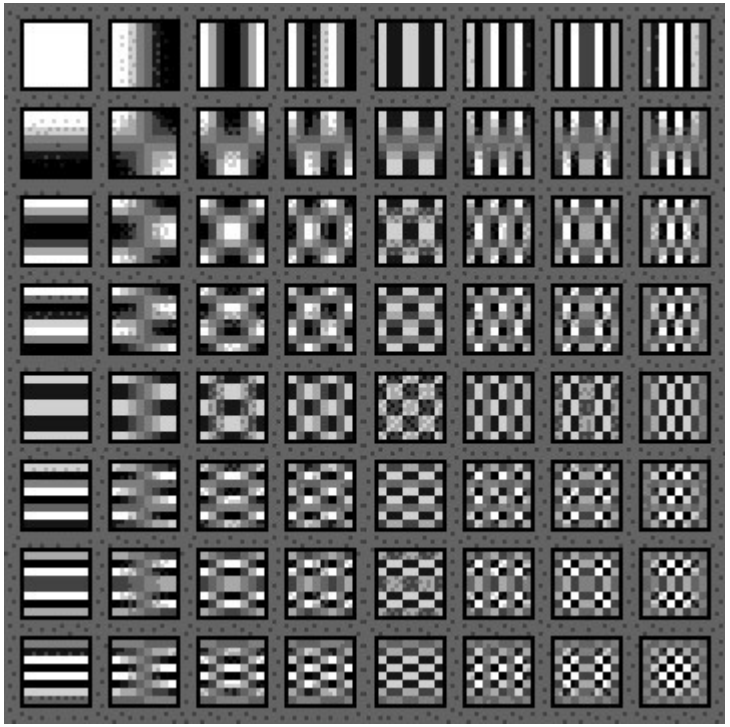
$$A(k) = \begin{cases} \frac{1}{\sqrt{2}}, k = 0 \\ 1 \quad , k > 0 \end{cases}$$

V(x,y) – компонента цвета (Y, Cr, Cb)

ДКП

$$C(i,j) = \frac{A(i)A(j)}{\sqrt{2N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} V(x,y) \cdot \cos\left(\frac{\pi i(2x+1)}{2N}\right) \cos\left(\frac{\pi j(2y+1)}{2N}\right)$$

свертку с таким изображением нужно сделать, чтобы получить соответствующий коэффициент ДКП

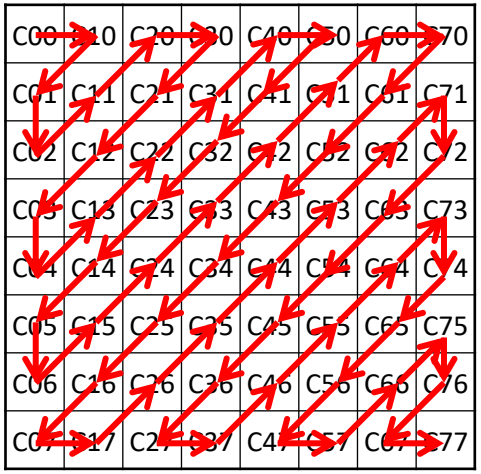


Образ базовых функций ДКП 8x8

Чем ниже и правее, тем выше частота и тем мельче детали на изображении

Кодирование JPEG

Шаг 5. Обход «зигзагом»
(получаем линейный массив из 64 чисел)



C00	C10	C20	C30	…	C57	C67	C77
AC0	DC1	DC2	DC3	…	DC61	DC62	DC63

Разрядность чисел минимум 12 бит !!!

Шаг 6. Квантизация делением на вектор «качества»
(получаем целые числа 8 бит)

n = целая часть от (DC/k)

k0 = 1 !!!

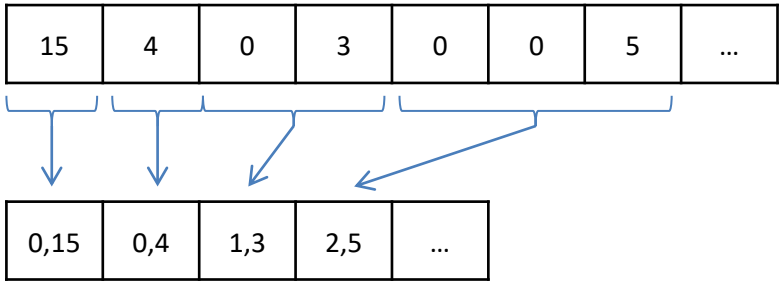
AC0	DC1	DC2	DC3	…	DC61	DC62	DC63
k0	k1	k2	k3	…	k61	k62	k63

↓

n0	n1	n2	n3	…	n61	n62	n63

Кодирование JPEG

Шаг 7. RLE-кодирование нулевых значений
получаем пары {Nz, Vnz} – Nz – число нулей, Vnz – ненулевое значение



Шаг 7. Кодирование Хаффмана полученных пар
Алфавитом являются все возможные пары {Nz, Vnz}

Реально таблица Хаффмана строится по парам {Nz, Anz},
где Anz – число бит у ненулевого значения
В файл записываются код из таблицы Хаффмана и все биты ненулевого значения.

Есть стандартная таблица Хаффмана (определена в стандарте)
или можно построить таблицу под конкретное изображение

Декодирование JPEG

Декодирование выполняется в обратном порядке:

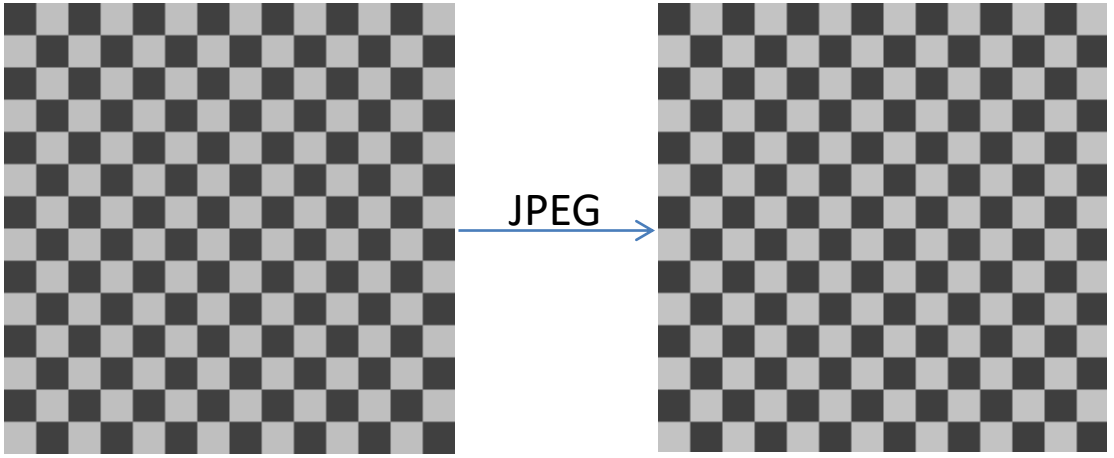
1. Восстановление из битового потока коэффициентов ДКП.
2. Умножение на вектор качества.
3. Обратный «зигзаг» (в двумерный массив)
4. Обратное двумерное ДКП
5. Сборка всех плоскостей Y, Cr и Cb
6. Интерполяция Cr и Cb до полного размера
7. Матрицирование Y, Cr и Cb в R, G и B

Вопросы к JPEG

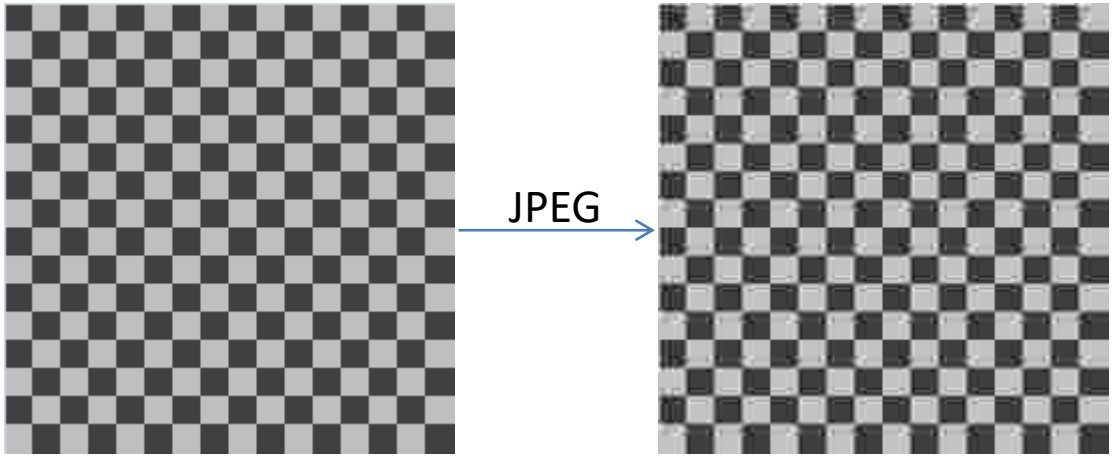
- Что влияет на степень сжатия?
- Что нужно передавать от кодера к декодеру?
- Как построить вектор «качества»? (какие ограничения накладываются)

Искажения JPEG

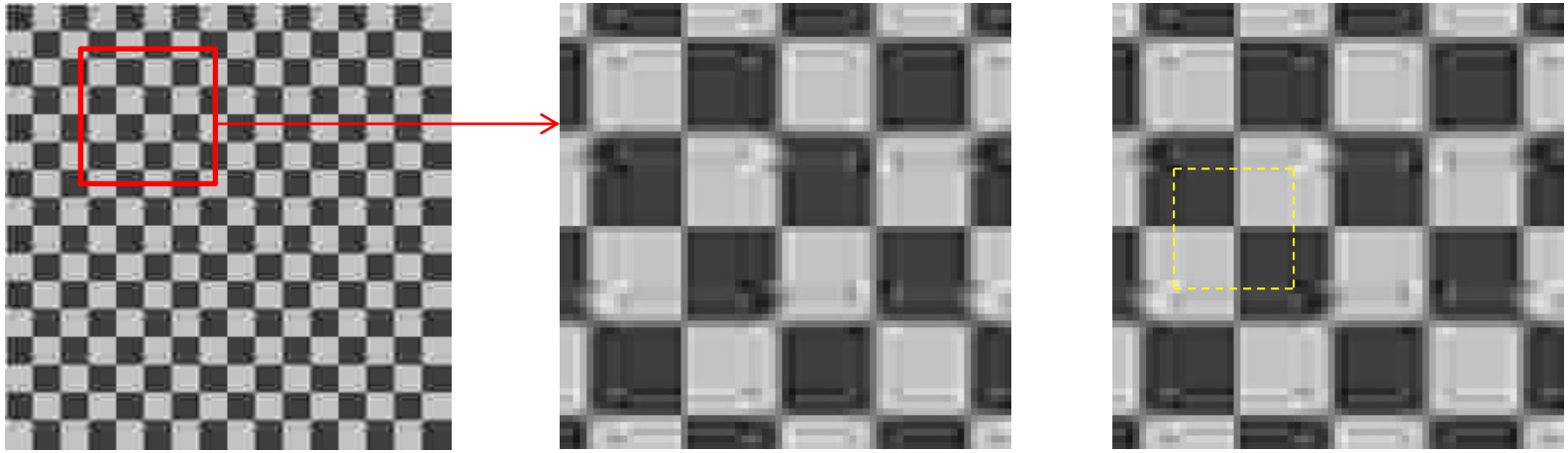
Шахматная доска
16x16



Шахматная доска
15x15

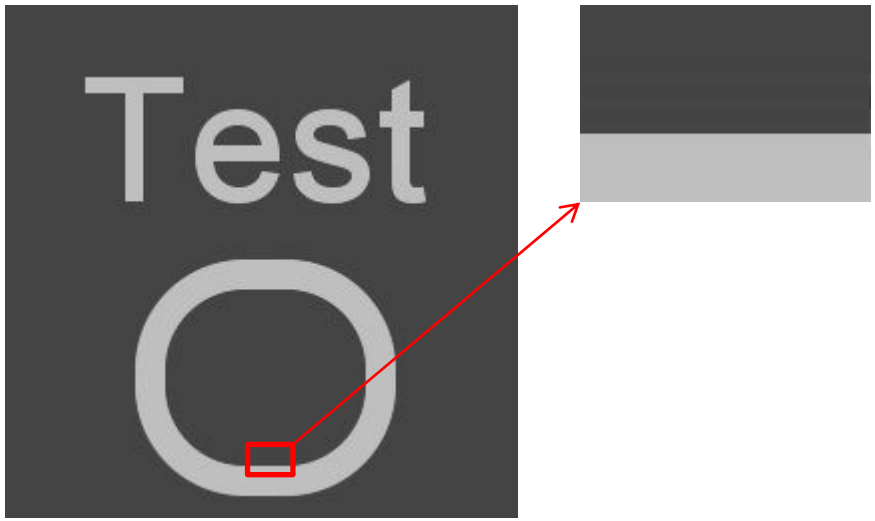


Искажения JPEG

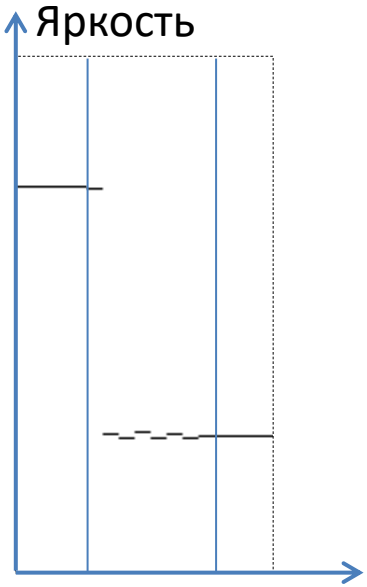


Блочность по границам квадратов 8x8

Искажения JPEG

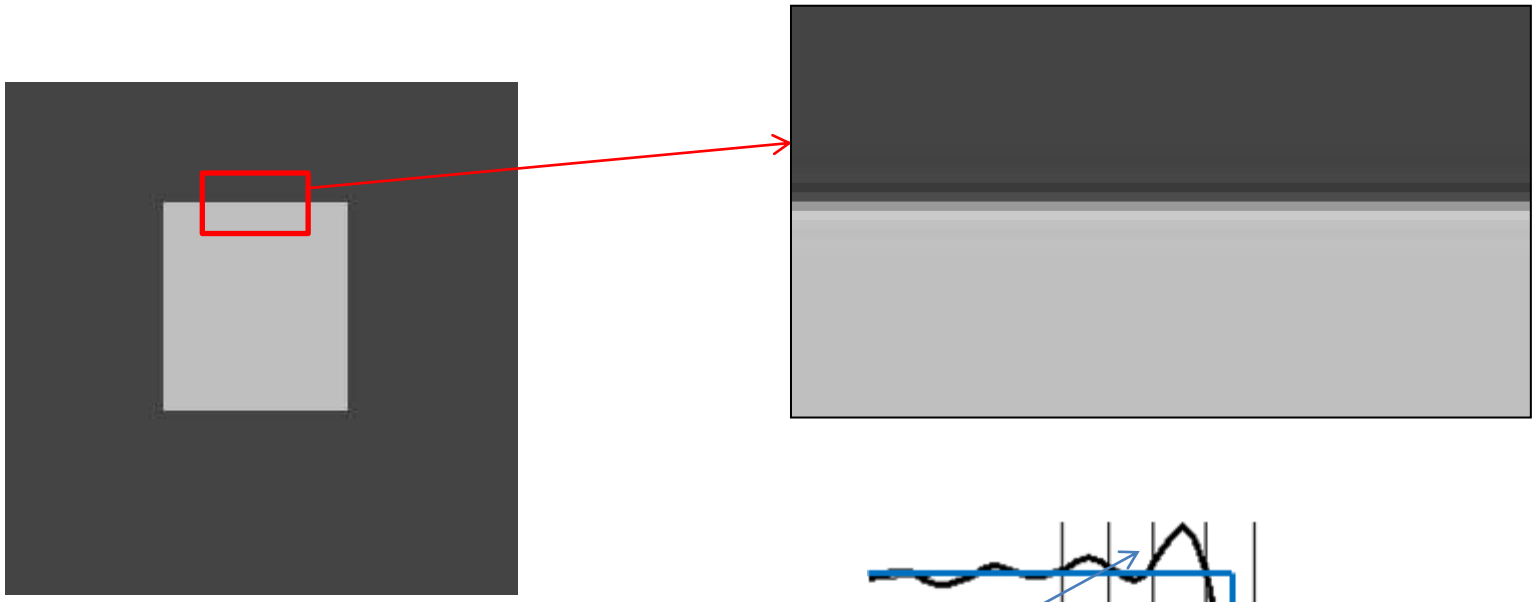


Ложные контура из-за эффекта Гиббса

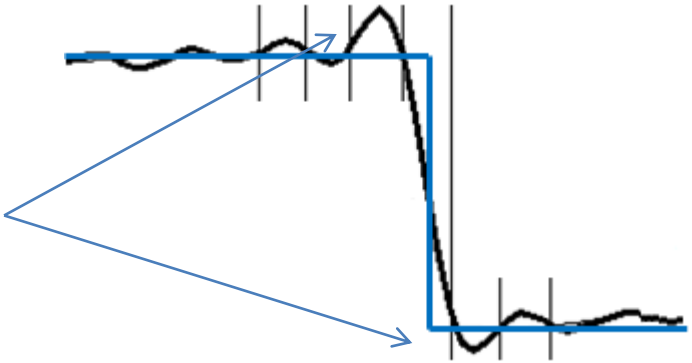


Координата по вертикали
(снизу вверх)

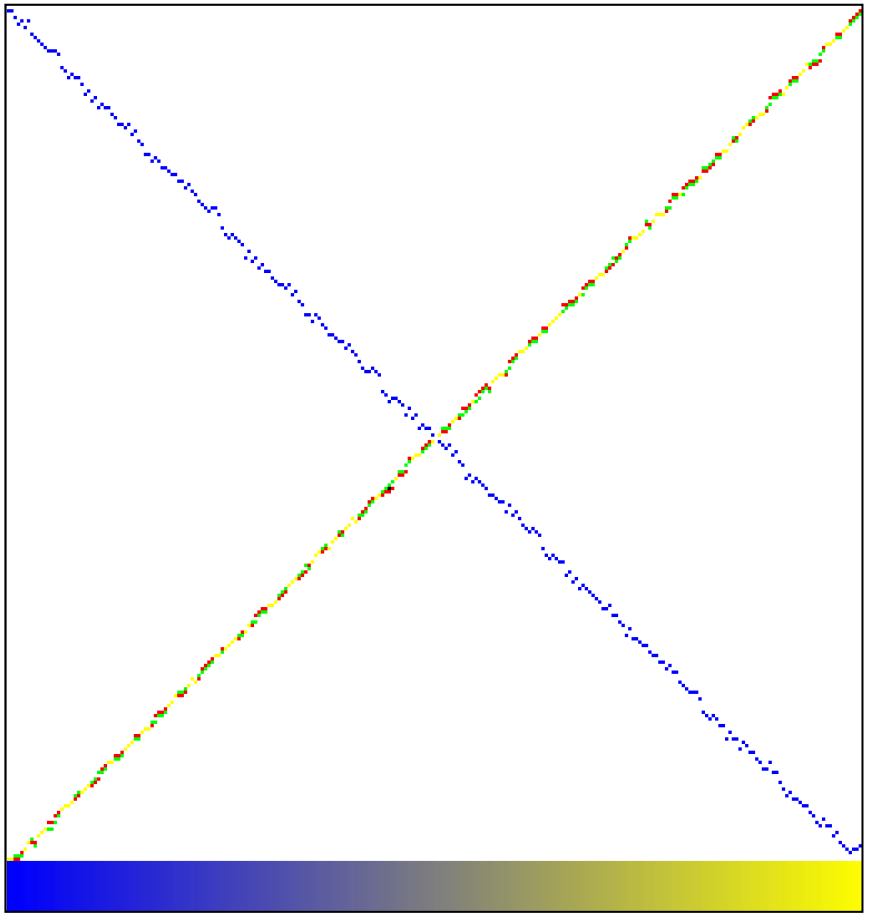
Искажения JPEG



Ложные контура из-за эффекта Гиббса



Искажения JPEG



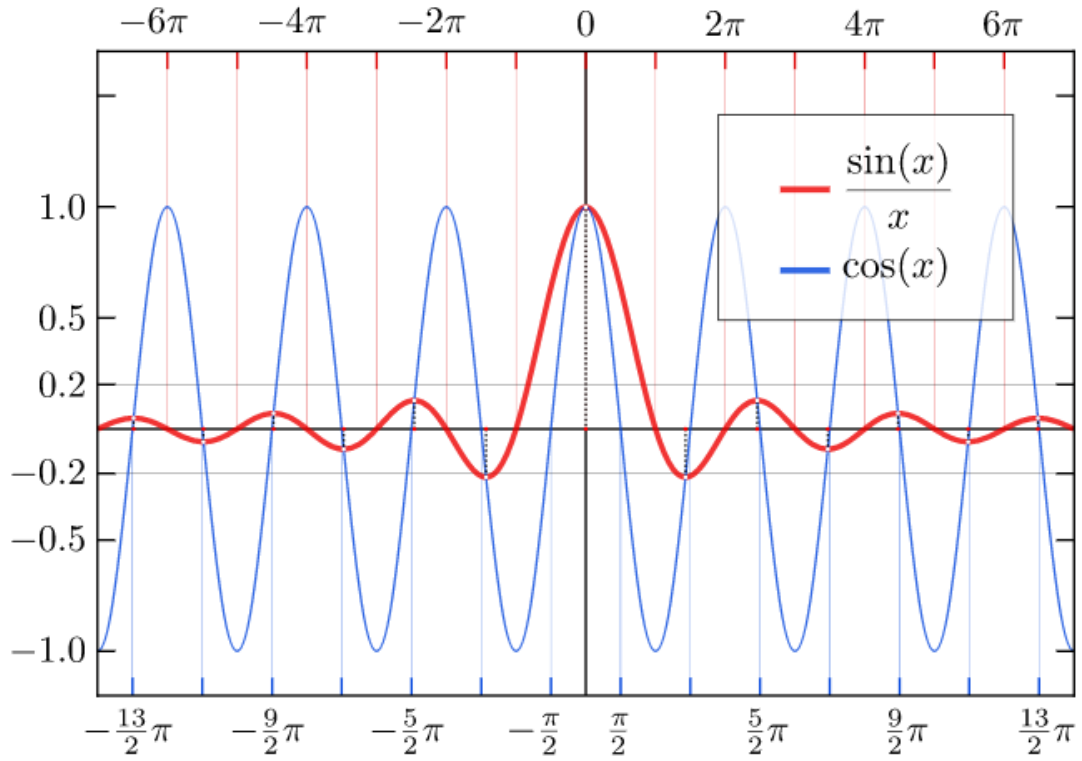
Из-за аналогичного эффекта на линейных градиентах возникают небольшие колебания, воспринимаемые зрителем как текстурное свойство объекта, не существующее на самом деле

Wavelet (JPEG-2000)

- «Всплеск» или «волновой пакет»
- Базовые функции локальны в пространстве (спадают до нуля на расстоянии от центра)
- Полный набор базовых функций получается из пары базовых функций посредством их сдвигов и растяжений по оси времени/пространства

- Характерный пример: $\operatorname{sync}(x) = \frac{\sin(x)}{x}$

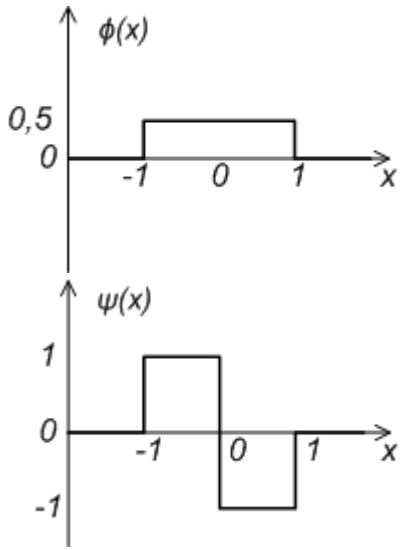
Wavelet



Базовая функция локальна во времени => искажение в одном месте
не влияет на другие места + свертка может делаться со всем изображением
(размер фильтра ограничен в пространстве небольшим окном)

Wavelet

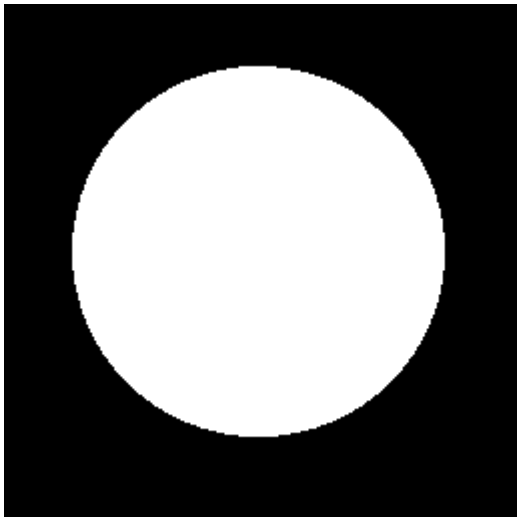
- Функции Хаара – простейший wavelet
- Интегрирующая функция ϕ(x)
  низкие частоты
- Дифференцирующая функция ψ(x)
  высокие частоты



- Ширина окна – 2 пикселя
- Для каждой пары значений (пикселей) получается один низкочастотный и один высокочастотный коэффициент
  из которых однозначно восстанавливается исходная пара значений

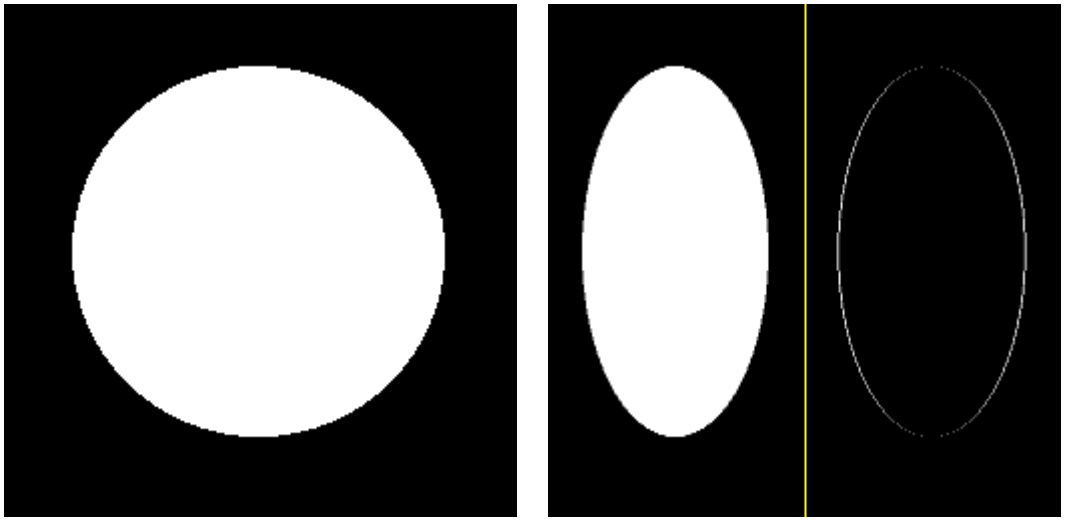
Кодирование Wavelet

- Преобразование в Y, Cr, Cb и децимация (прореживание) Cr и Cb (как в JPEG)



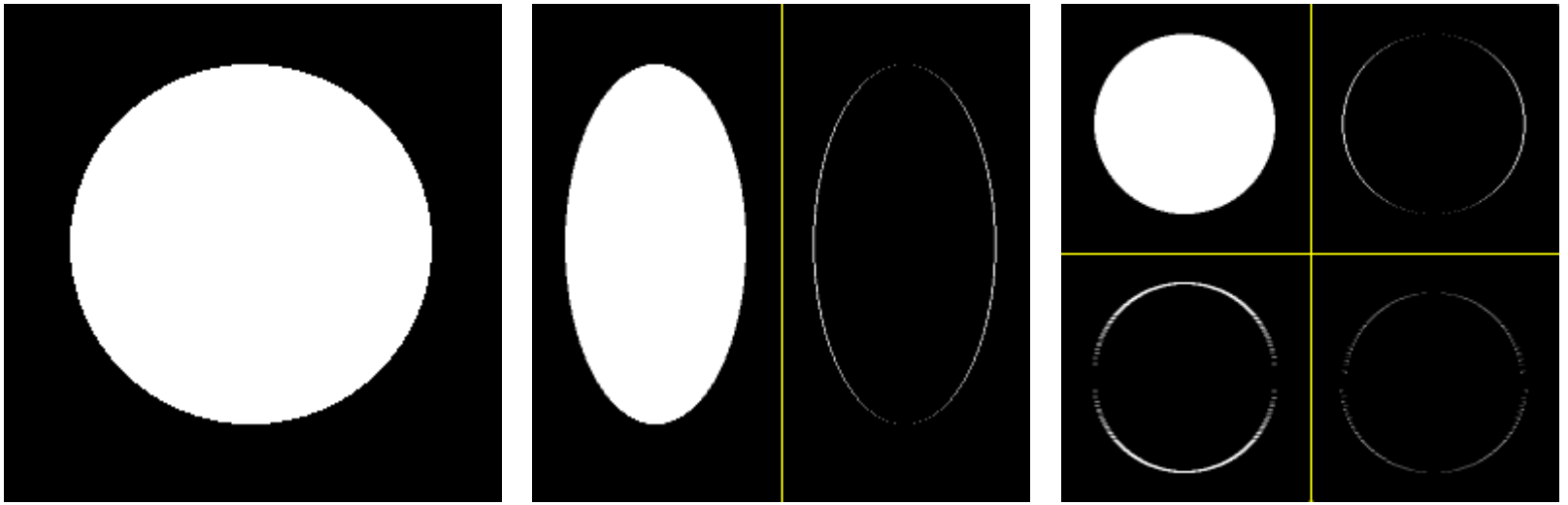
Кодирование Wavelet

- Преобразование в Y, Cr, Cb и децимация (прореживание) Cr и Cb (как в JPEG)
- Wavelet преобразование по горизонтали



Кодирование Wavelet

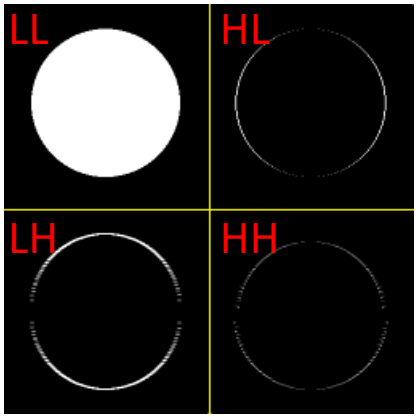
- Преобразование в Y, Cr, Cb и децимация (прореживание) Cr и Cb (как в JPEG)
- Wavelet преобразование по горизонтали
- Wavelet преобразование по вертикали



Кодирование Wavelet

На первом шаге из исходного изображения получили 4 набора коэффициентов:

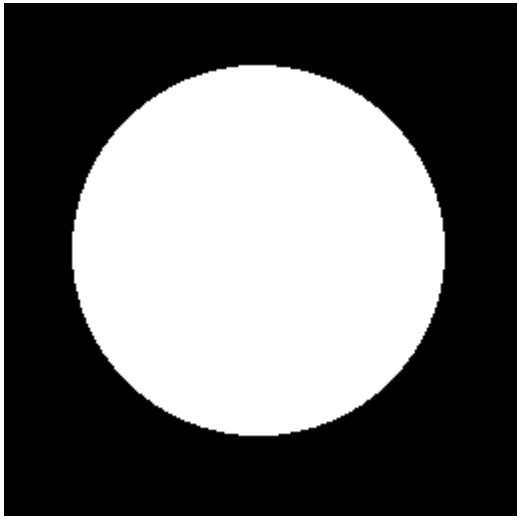
- В левом верхнем углу компоненты LL (немного размытое изображение)
- В правом вернем углу компоненты HL (отличия по горизонтали)
- В левом нижнем углу компоненты LH (отличия по вертикали)
- В правом нижнем углу компоненты HH (отличия по диагоналям)

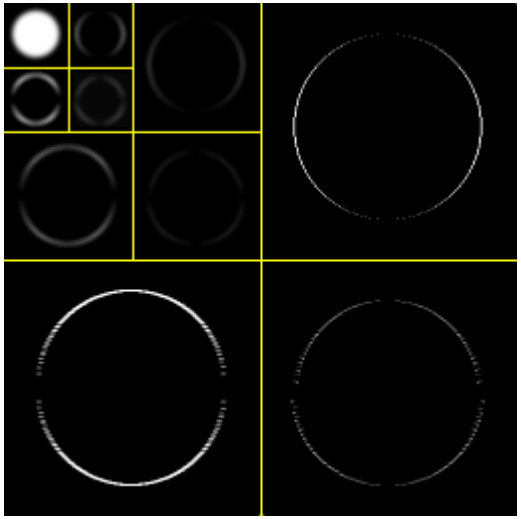


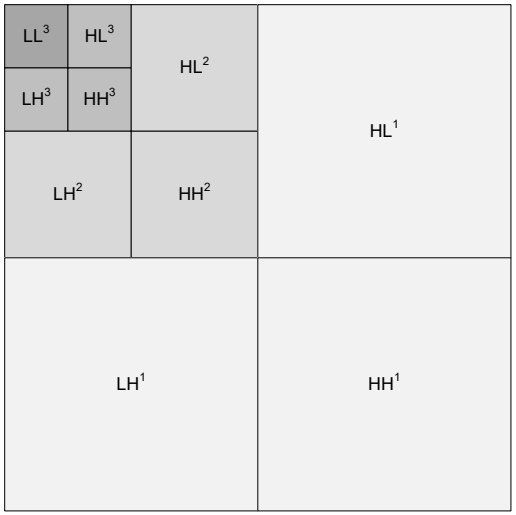
- Наборы HL, LH и HH содержат очень много нулей
- Набор LL содержит немного смазанное изображение его можно еще раз обработать тем же способом !!!

Кодирование Wavelet

После трех повторений получаем 3+3+3+1=10 наборов коэффициентов:







Для изображений большого разрешения
можно делать больше циклов кодирования

Кодирование Wavelet

- На изображении одному коэффициенту LL^3 соответствует по одному коэффициенту HL^3, LH^3, HH^3, которые все вместе получены из 4-х коэффициентов LL^2.
- Аналогично 4-м коэффициентам LL^2 соответствуют по 4-е коэффициента HL^2, LH^2, HH^2, (вместе – 16 коэффициентов LL^1).
- Аналогично 16-и коэффициентам LL^1 соответствуют по 16-й коэффициентов HL^1, LH^1, HH^1, (итого – 64 значения/пиксела).
- Таким образом можно сформировать группу из 64 коэффициентов, соответствующих одной группе ДКП 8x8.
- Либо можно передавать коэффициенты последовательно – сначала все LL^3, затем все HL^3, LH^3, HH^3, затем все HL^2, LH^2, HH^2, затем все HL^1, LH^1, HH^1. Это удобнее при скачивании по сети – из всех LL^3 уже можно понять, что за изображение передается.

Отличия Wavelet от JPEG

- Базовые функции Wavelet ограничены в пространстве, поэтому можно обрабатывать всю картинку (не делить ее на блоки) => нет эффекта блочности (сложность алгоритма сравнима с ДКП).
- Потеря высокочастотных коэффициентов оставляет низкочастотные коэффициенты, которые по своей сути аналогичны эффекту сглаживания или дефокусировки – для человека потеря четкости в изображении является привычным дефектом, поэтому такие искажения не вызывают сильного дискомфорта, аналогичному эффекту блочности или ложных контуров в JPEG-е.