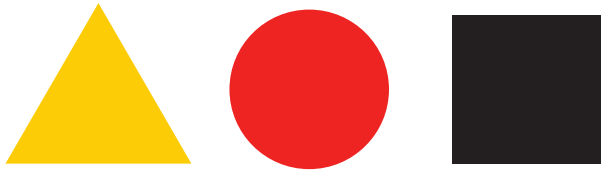


ШКОЛА АНАЛИЗА ДАННЫХ


Jpeg-decoder

Форматы

- Sequential DCT-based (baseline sequential) — нужно реализовать в ДЗ
- Progressive DCT based — для бонусов
- Sequential lossless (!= quality: 100%)
- Hierarchical

План

- Самый простой вариант baseline
- Chroma downsampling
- Особенности реализации
- Progressive

Baseline

Baseline encoding

- Работаем с 8-битными R'G'B'
- Штрихи важны!
- 1 шаг — R'G'B' -> Y'CbCr
- $Y' = 0.299R' + 0.587G' + 0.114B'$
- Y' — яркостная составляющая; Cb, Cr — хроматическая
- 2 шаг — разбиваем изображение на блоки 8x8 (дублируем последний столблец/строку для кратности)

Baseline

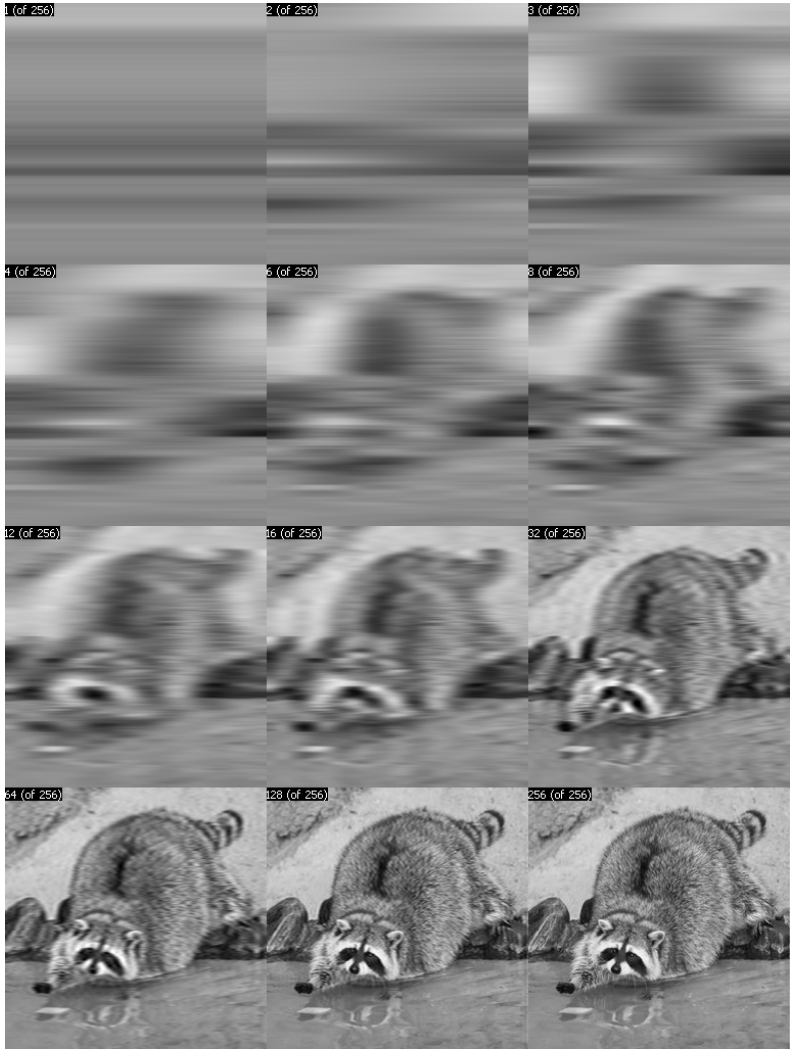
- Каждый блок каждой компоненты обрабатывается одинаково, рассмотрим на примере Y'
- Отнимаем от каждого элемента 128, вычисляем двумерное дискретное косинусное преобразование

$$G(u,v) = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right)$$

- $\alpha(u) = \frac{1}{\sqrt{2}}$ при $u = 0$ и 1 иначе.

Baseline

- $G(0, 0)$ — DC-коэффициент (почти среднее всех значений в блоке), остальное — AC-коэффициенты
- Коэффициенты с маленькими u, v содержат низкие частоты, дальше — высокие (грубо говоря, сначала основной тон, потом мелкие детали)
- Низкочастотные коэффициенты важнее


1 (of 256)
2 (of 256)
3 (of 256)
4 (of 256)
8 (of 256)
8 (of 256)
12 (of 256)
16 (of 256)
32 (of 256)
64 (of 256)
128 (of 256)
256 (of 256)

Baseline

- Делим почленно получившуюся после DCT матрицу на таблицу квантования и округляем до целого

a. Low compression

1	1	1	1	1	2	2	4
1	1	1	1	1	2	2	4
1	1	1	1	2	2	2	4
1	1	1	1	2	2	4	8
1	1	2	2	2	2	4	8
2	2	2	2	2	4	8	8
2	2	2	4	4	8	8	16
4	4	4	4	8	8	16	16

b. High compression

1	2	4	8	16	32	64	128
2	4	4	8	16	32	64	128
4	4	8	16	32	64	128	128
8	8	16	32	64	128	128	256
16	16	32	64	128	128	256	256
32	32	64	128	128	256	256	256
64	64	128	128	256	256	256	256
128	128	128	256	256	256	256	256

- Основной этап, когда теряется качество

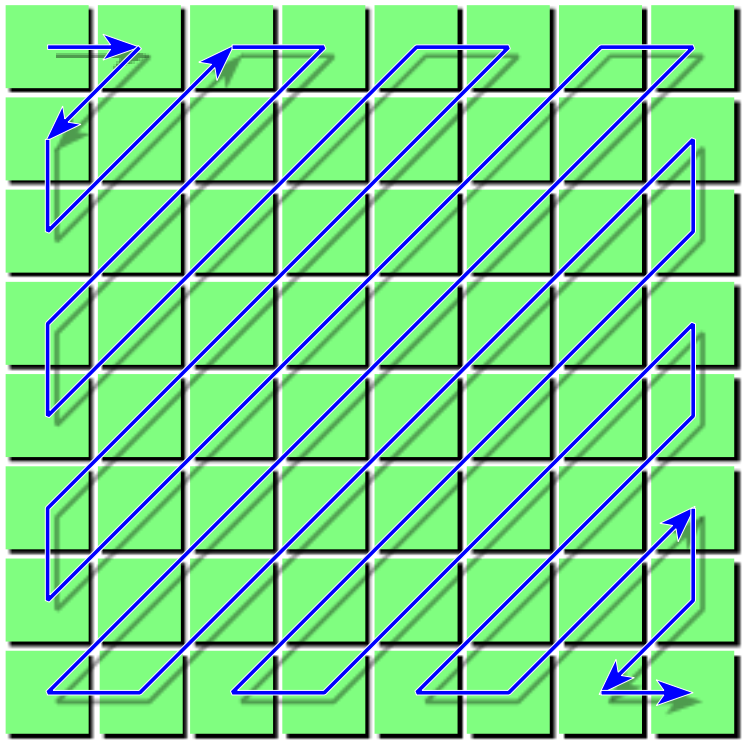
Baseline

- Зато в высоких частотах теперь много нулей

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Baseline

- Выписываем матрицу в вектор с помощью zig-zag обхода



Baseline

- Каждый коэффициент вектора кодируется так: пишется число нулей до него, потом сам коэффициент. Пара (0, 0) означает, что все до конца вектора с этого момента заполнено нулями.
- Например, вектор (4, 0, 0, 3, 6, 19, 1, 0, 5, 0, ..., 0) закодируется как (0, 4), (2, 3), (0, 6), (0, 19), (0, 1), (1, 5), (0, 0)

Baseline

- Полученный список пар специальным образом кодируется и сжимается кодом Хаффмана
- Код общий для всех блоков заданной компоненты

Baseline

- Описанная процедура выполняется для всех блоков всех компонент
- Полученные закодированные блоки пишутся в файл в порядке сверху вниз, слева направо, чередуя компоненты. Т.е. $Y_{0,0}, Cb_{0,0}, Cr_{0,0}, Y_{0,1}$ и т.д.
- В декодере (который и нужно написать) все в обратном порядке.

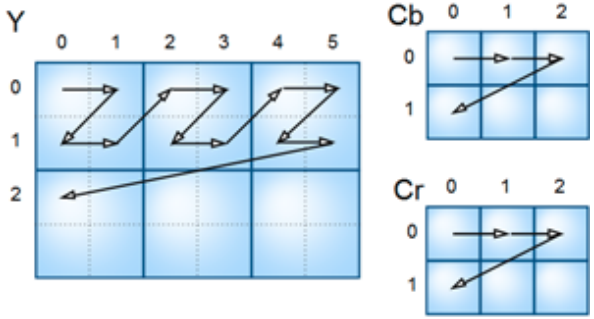
Chroma downsampling

Chroma downsampling

- Еще одно место, где теряется качество, выполняется сразу после конвертации R'G'B' -> Y'CbCr
- Хроматические составляющие менее важны, поэтому их можно частично выкинуть. Например, заменим каждый блок 2x2 в cb и cr на среднее в этом блоке
- 4 режима
  - 4:4:4 — нет сабсемплинга
  - 4:2:2 horizontal — усредняем каждый блок 1x2
  - 4:2:2 vertical — усредняем каждый блок 2x1
  - 4:2:0 — усредняем каждый блок 2x2

Chroma downsampling

- Понятие блока остается тем же — матрица 8x8, вводится понятие MCU — блок реального исходного изображения
- Например, в 4:2:0 размер MCU 16x16, он содержит 4 блока Y и по 1 блоку Cb и Cr
- В файл по порядку сверху вниз, слева направо пишутся MCU. Внутри самого MCU сначала пишутся сверху вниз, слева направо блоки Y, потом Cb и Cr



Особенности реализации

Требования к коду

- IDCT писать не руками, а использовать http://www.fftw.org/
- Декодирование кода Хаффмана нужно вынести в отдельный класс, который ничего о jpg не знает.
- Сам декодер стоит оформить в виде класса, принимающего `std::istream&`
- Декодер не должен падать ни на каких входных данных (проверять все на валидность и кидать исключения при случае)

Особенности реализации

- Открыть файл в бинарном режиме `std::ifstream stream("foo.txt", ios_base::binary);`
- Например, так можно прочитать 10 байт:

```
char bytes[10];
stream.read(bytes, 10);
```

- Не забываем проверять поток на .eof()
- В jpeg используется big-endian, поэтому собирать 2-байтовое слово в число стоит вручную
- Иногда нужно будет считывать побитово, эту логику стоит инкапсулировать в отде класс (типа BitReader), который помнит последний считанный байт

Структура хедера

- [маркер][длина][контент]
- Маркер имеет вид ff ?? (например ffd8). Декодер должен поддерживать маркеры SOI, SOF0, DHT, DQT, APPn, COM, EOI, SOS. Progressive-декодер также SOF2.
- Длина — задается словом, для всех хедеров, кроме SOS, включает в себя само это слово и контент.
- Например, комментарий hello кодируется в jpeg набором байт ff fe 00 07 68 65 6c 6c 6f

Хедер DHT

- Маркер ffc4
- В контенте сначала пишется байт xy, где x = 0, если это таблица DC коэффициентов, и x = 1 для AC. y — id таблицы.
- Далее следует 16 байт, i-ый байт содержит количество кодов длины i в данном дереве Хаффмана. Этой информации достаточно, чтобы восстановить дерево. Алгоритм: https://www.impulseadventure.com/photo/jpeg-huffman-coding.html
- Затем перечислено содержимое дерева — сначала 1-байтовые коды, потом 2-байтовые и т.д.

Хедер SOS

- Собственно, сами MCU в описанном ранее порядке. В baseline sos должен быть только 1, в progressive их несколько.
- Маркер ffda
- Длина включает в себя только заголовочную часть, не само содержимое.
- Далее следует описание компонент (какие таблицы квантования и ht используются), после чего сами данные
- Данные считываем, пока не наткнемся на очередной маркер
- Стоит отметить, что 2 байта ff ff в этой секции на самом деле означают просто байт ff.

Как закодированы коэффициенты

- Вернемся к списку пар. Первая особенность — первый элемент пары не больше 15, т.е. пара (18, 7) будет разбита на 2 пары (15, 0), (2, 7)
- На примере (2, 6). Она будет записана как байт 23, после чего будет 3 бита 110. В байте 23 старший полубайт (2) содержит количество предыдущих нулей, в младшем полубайте (3) записана длина коэффициента в битах. Далее следуют эти 3 бита.
- Выбирается минимальное число бит, чтобы закодировать коэффициент. Например, для 13 это 4 бита: 1101. Если коэффициент отрицательный, то берется код его модуля и инвертируется, например -13 будет записан как 0010.
- Хаффманом закодированы именно описанные выше байты, биты коэффициента пишутся как есть.

AC/DC

- Случай 0 обрабатывается отдельно
- Для DC коэффициента будет записан просто один байт 00
- Для AC коэффициентов, если это случай (15, 0), как в примере выше, то эта пара будет закодирована как f0, а если это случай (0, 0), то просто одним байтом 00. Других случаев быть не может.
- В DC коэффициенте записан не реальный коэффициент, а та прибавка, которую нужно добавить к DC (реальному!) предыдущего (в порядке обхода) блока

AC/DC

- Случай 0 обрабатывается отдельно
- Для DC коэффициента будет записан просто один байт 00
- Для AC коэффициентов, если это случай (15, 0), как в примере выше, то эта пара будет закодирована как f0, а если это случай (0, 0), то просто одним байтом 00. Других случаев быть не может.
- В DC коэффициенте записан не реальный коэффициент, а та прибавка, которую нужно добавить к DC (реальному!) предыдущего (в порядке обхода) блока

Progressive jpeg

Progressive

- Теперь сканов может быть несколько — каждый следующий скан улучшает качество изображения, полезно при медленной скорости соединения. Хорошие progressive кодеры сжимают большие изображения сильнее, чем baseline при том же качестве.
- 2 приема: spectral selection и successive approximation
- Spectral selection — пишем не все 64 коэффициента блока, а только выбранный диапазон, например с 5 по 20.
- Successive approximation — сначала пишем старшие байты коэффициентов, в следующих сканах добавляем младшие
- В SOS в самом конце заголовочной части есть 3 байта, которые в baseline всегда должны быть равны 00 3f 00. Это на самом деле описание очередного скана в progressive — первые 2 байта содержат диапазон spectral selection (в baseline всегда 0-63), последний байт содержит информацию о successive approximation

Spectral selection

- Первым всегда идет DC-скан (т.е. диапазон 0-0), по стандарту он должен быть отделен от AC. Кодируются только DC-коэффициенты всех компонент
- Далее следуют AC-сканы. Каждый такой скан содержит только одну компоненту.

Successive approximation

- Обычно используется вместе с предыдущим методом вместе. Пусть последний байт в заголовочной части равен hl. В первом скане для заданного диапазона коэффициентов пишутся значения вида $\frac{x}{2^l}$.

  Последующие сканы будут добавлять ровно по 1 биту, куда именно добавляется бит записано в h.

- Способ кодирования всего этого отличается от baseline, разбирайтесь со спецификацией (удачи).