Тема 2.
Форматы хранения данных. Методы получения данных из веб и фотокамер.

Форматы хранения графических данных. BitMAp (BMP)

- Растровый формат, разработка Microsoft. Глубина цвета 1,4,8,18,24,32,48 бит.
- Если глубина цвета меньше 16 бит, хранится палитра RGB цветов (индексная таблица), а в поле данных указывается номер индекса.
- Хранит только один слой

  Обычно bmp — несжатый массив данных.

Lena, глубина цвета 8, 4 бит, 2 бит

TIFF

- Tagged Image Format — формат хранения растровых данных с большой глубиной цвета. Первоначально разработан Aldus Corporation(в дальнейшем слилась с Adobe) в кооперации с Microsoft.
- возможность применения сжатия без потерь
- возможность применения сжатия с потерями
- поддерживает разнообразные палитры
- Хранит несколько слоев изображения
- Поддерживает привязку к другим информационным слоям,например гео-привязку.

GIF

- Graphics Interchange Format (Разработка Compuserv.gif)

  Может отображать не более 256 оттенков цветов (размер индексной палитры жестко фиксирован)

  Не уменьшает детальность изображения (LZW сжатие) — без потерь.
- Стандарт GIF89a модифицирован с учетом прозрачности и анимации.

  Первоначально GIF использовал проприетарные алгоритмы, но срок последнего IBM патента истек в 2006 году, тем не менее патент на LZW был в в силе до недавнего времени,что ограничивало применение данного формата в GNU/GPL проектах
- Хранит только один слой.

PNG — свободная замена GIF

История возникновения- необходимость иметь свободный формат, по свойствам не хуже,чем GIF

- Полутоновое изображение до 16 бит
- Цветное индексированное изображение(8 бит , цвет 24 бит)
- Полноцветное изображение (до 48 бит)

- PNG- обеспечивает «реальное» сжатие без потерь с глубиной до 48 бит.
-
- Сжатие Deflate+ LZ27 (LZW у GIF)
- http://www.artlebedev.ru/tools/technogrette/img/png-1/ - почитать тут.
- http://www.artlebedev.ru/tools/technogrette/img/png-2/
- http://www.artlebedev.ru/tools/technogrette/img/png-3/
-
-

Сжатие с потерями

- Сжатие без потерь: преобразования обратимы.
- Сжатие с потерями а) цветовая информация- постеризация
- Сжатие с потерями б) искажающее сжатие

  главная задача сжатия с потерями- искажение данных несущественны с точки зрения их дальнейшего применения.

Краткий экскурс в алгоритмы сжатия данных, наиболее часто используемых в IT

1.RLE
2. LZ
3. Метод Хаффмана

Run Length Encoding (RLE)

- Один из самых простых и самых старых алгоритмов
- Одна из реализаций алгоритма такова: ищут наименее часто встречающийся байт, называют его префиксом и делают замены цепочек одинаковых символов на тройки

  "префикс, счетчик, значение".
- Если же этот байт встречается в исходном файле один или два раза подряд, то его заменяют на пару "префикс, 1" или "префикс, 2". Остается одна неиспользованная пара "префикс, 0", которую можно использовать как признак конца упакованных данных.
- Алгоритм применяется в форматах PCX, TIFF, BMP.

RLE

Простейший пример:

RRRRRRRBGGGGGGGGRGGGG

{esc}07R{esc}01B{esc}07G{esc}01R{esc}04G

Работает хорошо только там,где есть длинные монотонные последовательности _байт_.

Алгоритмы семейства LZ

- Авторство — Jacob Ziv& Abraham Lempel 1977
- База для PKZIP,LHA,ARJ,GIF....
- Идея: создать словарь часто повторяющихся последовательностей и использовать ссылки на них.

- LZ Использует уже просмотренную часть сообщения как словарь,пытаясь заменить очередной фрагмент на указатель.
- Скользящее по сообщению окно, состоящее из двух частей: просмотренное сообщение и буфер.

for (i=0; i<LIM-1; i++)\r for (j=i+I; j	<MAX; j++)\r
Обработанная часть сообщения (словарь)	Буфер

- Результат коды из 3х элементов:
  - Смещение в словаре относительно начала подстроки, совпадающей с содержимым буфера
  - Длина
  - Первый символ в буфере,следующий за подстрокой

Мы вольные птицы; пора, брат, пора,<o15,4,!>!

Туда, где за тучей белеет гора,

[Туда, где]02 синеют морские края,

[Туда, где]02 гуляем лишь ветер... да я!.." (А.С. Пушкин)

- При нахождении известной подстроки в словаре, алгоритм сдвигает все содержимое подстроки на +1 символ влево и втягивает очередное соотв. Количество новых сообщений
- Если ничего нет, то <0,0,первый символ буфера> и далее.
- Хотя такое решение не оптимально (нужно много циклов) кодирование гарантировано.

Упрощенный LZ77

УПРОЩЕННЫЙ АЛГОРИТМ LZ77

- пока (буфер не пуст)
  найти наиболее длинное соответствие (позиция_начала, длина)
  в обработанной_части_сообщения и в буфере;
- если (длина > минимальной_длины), то
  вывести в кодированные данные пару (позиция _начала, длина);
- иначе
- вывести в кодированные данные первый символ буфера;
- изменить указатель на начало буфера и продолжить.

LZ77

Содержимое окна	Содержимое буфера	КОД
$kababababab z$	k	<0,0,k>
$kababababab z$	a	<0,0,a>
$kababababab z$	b	<0,0,b>
k ab $ababababz$	aba	<2,2,a>
$kaba$ babab abz	$bababz$	<2,5,z>

Проблемы классического LZ77

- Степень сжатия зависит от длины окна(4K/~64), что уменьшает скорость работы.
- Декодирование намного быстрее кодирования, но скорость декодирования практически не зависит от степени сжатия
- Необходимость в сдвигах больших объемов памяти
- Общего словаря нет- можно сослаться только в пределах текущего положения окна.
- Проблемы кодирования данных большой энтропии: вместо сжатия получаем увеличение объема до 3х раз (<0,0,символ>)

LZ78

- Развитие идеи LZ77, работает на другом принципе.
- Работает по принципу «жадности», вносит все вновь полученные фразы в словарь
- если последовательность находится в словаре, то на выход подается код для этой записи
- если последовательность — дополненная версия последовательности из словаря, то эта последовательность добавляется в словарь
-

  Каждый раз, когда программа встречает последовательность из словаря, она выдает код и добавляет в словарь новую последовательность, которая на один байт длиннее.

LZ78 kabababababz

Содержимое словаря	Содержимое считываемой строки	КОД
	k	<0,k>
k	a	<0,a>
k, a	b	<0,b>
k, a, b	ab	<2,b>
k, a, b, ab	aba	<4,a>
k, a, b, ab, aba	ba	<3,a>
k, a, b, ab, aba, ba	$abab$	<5,b>
k, a, b, ab, aba, ba, $abab$	z	<0,z>

LZW- (Lempel-Ziv-Welch).

LZW-сжатие заменяет строки символов кодами. Это делается без какого-либо анализа входного текста. Вместо этого при добавлении каждой новой строки символов просматривается таблица строк. Сжатие происходит, когда код заменяет строку символов. Коды, генерируемые LZW-алгоритмом, могут быть любой длины, но они должны содержать больше бит, чем единичный символ. Первые 256 кодов (когда используются 8-битные символы) по умолчанию соответствуют стандартному набору символов. Остальные коды соответствуют обрабатываемым алгоритмом строкам.

LZW

Процедура LZW-сжатия

```
СТРОКА = очередной символ из входного потока
WHILE входной поток не пуст DO
  СИМВОЛ = очередной символ из входного потока
  IF СТРОКА+СИМВОЛ в таблице строк THEN
    СТРОКА = СТРОКА+СИМВОЛ
  ELSE
    вывести в выходной поток код для СТРОКА
    добавить в таблицу строк СТРОКА+СИМВОЛ
    СТРОКА = СИМВОЛ
  END of IF
END of WHILE
вывести в выходной поток код для СТРОКА
```

«abacabadabacabae».

- Шаг 1. заполнили 5 символов кода символами[0:a,1:b,2:c,3:d,4:e]. начали считали a
- Шаг 2. Считали b->ab → в таблицу 5:ab, в поток 0{a}.
- Шаг 3. a-> ba-> в таблицу 6:ba, в поток 1{b}
- Шаг 4. c-> ac → в таблицу 7:ac в поток 2{c}
- Шаг 5. a->ca → в таблицу 8:ca в поток 0{a]
- Шаг 6. b ab — есть в таблицу 9:aba в поток 5{ab}
- Шаг 7. d: “ad” — нет. В таблицу: <10> “ad”. В поток: <0>;
- Шаг 8: “ad” — нет. В таблицу: <10> “ad”. В поток: <0>;
- Шаг 9: “da” — нет. В таблицу: <11> “da”. В поток: <3>;
- Шаг 10 “aba” — есть в таблице; “abac” — нет. В таблицу: <12> “abac”. В поток: <9>;
- Шаг 11: “ca” — есть в таблице; “cab” — нет. В таблицу: <13> “cab”. В поток: <8>;
- Шаг 12: “ba” — есть в таблице; “bae” — нет. В таблицу: <14> “bae”. В поток: <6>;
- Шаг 13: И, наконец последняя строка “e”, за ней идет конец сообщения, поэтому мы просто выводим в поток <4>.
-
-

Текущая строка	Текущий символ	Следующий символ	Вывод		Словарь	
			Код	Биты		
ab	a	b	0	000	5:	ab
ba	b	a	1	001	6:	ba
ac	a	c	0	000	7:	ac
ca	c	a	2	010	8:	ca
ab	a	b	-	-	-	-
aba	b	a	5	101	9:	aba
ad	a	d	0	000	10:	ad
da	d	a	3	011	11:	da
ab	a	b	-	-	-	-
aba	b	a	-	-	-	-
abac	a	c	9	1001	12:	abac
ca	c	a	-	-	-	-
cab	a	b	8	1000	13:	cab
ba	b	a	-	-	-	-
bae	a	e	6	0110	14:	bae
e	e	-	4	0100	-	-

- Итого: 0 1 0 2 5 0 3 9 8 6 4 = 11 бит маржа

Данные		На выходе	Новая запись	
Биты	Код		Полная	Частичная
000	0	a	- -	5: a?
001	1	b	5: ab	6: b?
000	0	a	6: ba	7: a?
010	2	c	7: ac	8: c?
101	5	ab	8: ca	9: ab?
000	0	a	9: aba	10: a?
011	3	d	10: ad	11: d?
1001	9	aba	11: da	12: aba?
1000	8	ca	12: abac	13: ca?
0110	6	ba	13: cab	14: ba?
0100	4	e	14: bae	- -

Как быть с повторяющимися символами аааааааа?

-
- Итак, кодировщик заносит первую «а» в строку, ищет и находит «а» в словаре. Добавляет в строку следующую «а», находит, что «аа» нет в словаре. Тогда он добавляет запись <5>: «аа» в словарь и выводит метку <0> («а») в выходной поток.
- Далее строка инициализируется второй «а», то есть принимает вид «а?» вводится третья «а», строка вновь равна «аа», которая теперь имеется в словаре.
- Если появляется четвертая «а», то строка «аа?» равна «ааа», которой нет в словаре. Словарь пополняется этой строкой, а на выход идет метка <5> («аа»).
- После этого строка инициализируется третьей «а», и т.д. и т.п. Дальнейший процесс вполне ясен.
- В результате на выходе получаем последовательность 0, 5, 6 …, которая короче прямого кодирования стандартным методом LZW.
-

Оговорка при декодировании

- 0- A
- 5- кода еще нет. Но мы можем догадаться что это — aa
- 6 — кода тоже нет. Но по правилу это aaa.
- Итд, работает также для последовательностией.

Свойства LZW

- Эффективен там,где встречаются повторяющиеся структуры (изображения,видео,тексты)
- Не требует хранения таблиц словарей.
- Не требует вычисления значений вероятностей
- Чем выше «случайность данных» тем хуже сжатие.
-

Метод Хаффмана
Дэвид Хаффман(1925-1999,США)

- Метод сжатия на основе двоичных кодирующих деревьев (1952 год -аспирант MIT)
- Идея алгоритма: зная вероятность вхождения символов в сообщение, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью присваиваются более короткие коды. Коды Хаффмана имеют уникальный префикс, что и позволяет однозначно их декодировать, несмотря на их переменную длину. Динамический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана.
- Задачи: 1. Найти вероятность
- 2. Закодировать дерево
-

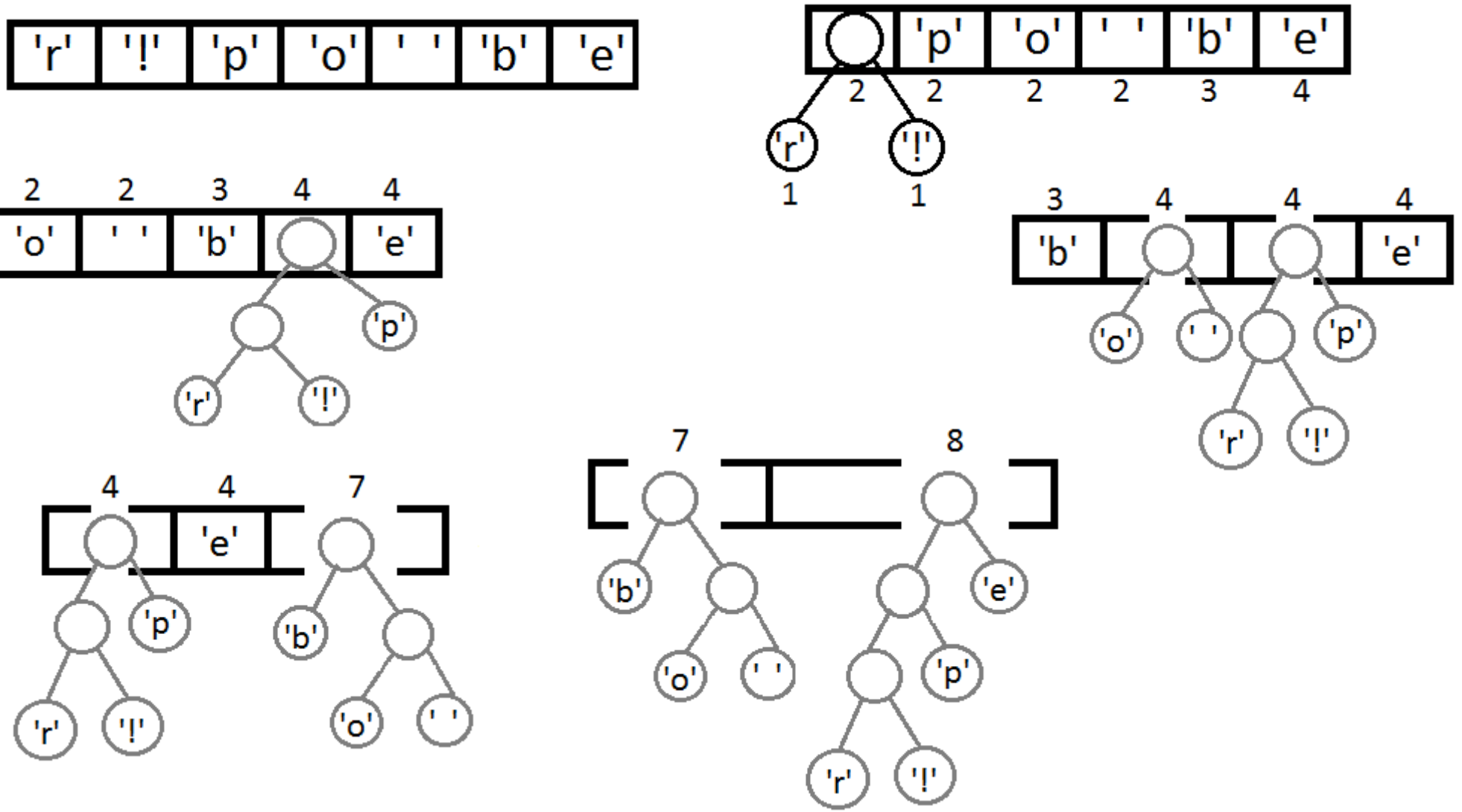
Метод Хаффмана

- Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в ожидаемое сообщение.
- Выбираются 2 свободных узла дерева с наименьшими весами.
- Создается родитель с весом равным их суммарному весу.
- Родитель добавляется в список свободных узлов, а двое его потомков удаляются из этого списка.
- Одной дуге выходящей их родителя ставится в соответствие бит 1, другой – бит 0.
- Далее пункты повторяются, начиная со второго, до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

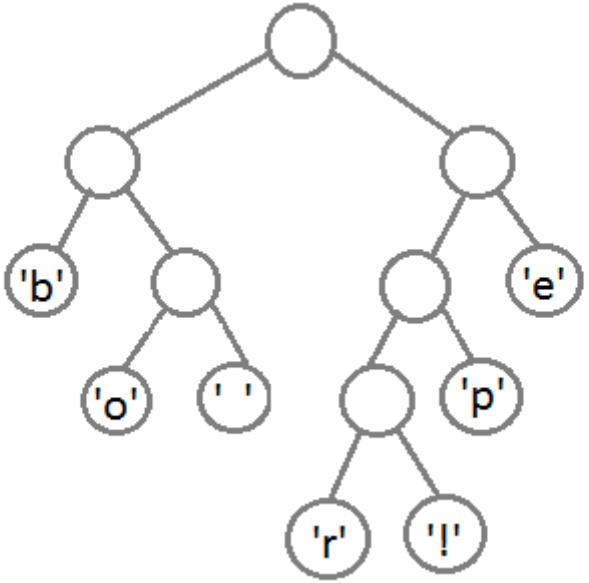
Пример

(http://habrahabr.ru/post/144200/)

«beep boop beer!» частоты b:3 e:4 p:2 ' ':2 'o':2 'r':1 '!':1

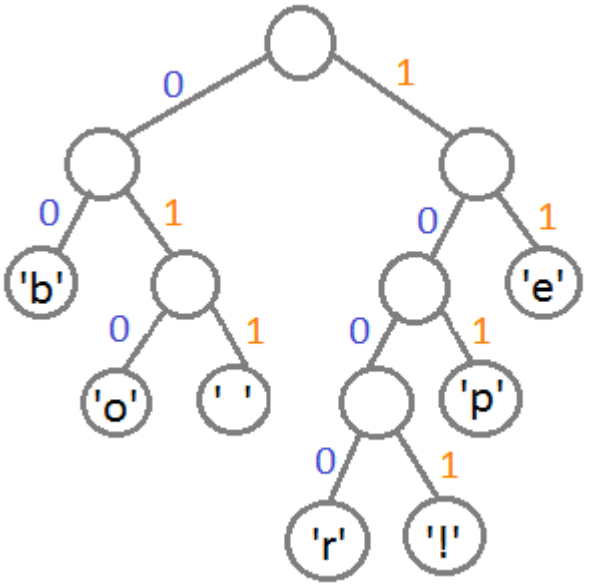


Итоговое дерево:



Кодирование-декодирование

- Налево пойдешь-1, направо 0



'b'	00
'e'	11
'p'	101
' '	011
'o'	010
'r'	1000
'!'	1001

Хаффман

- Важно иметь в виду, что каждый код не является префиксом для кода другого символа. В нашем примере, если 00 — это код для 'b', то 000 не может оказаться чьим-либо кодом, потому что иначе мы получим конфликт. Мы никогда не достигли бы этого символа в дереве, так как останавливались бы ещё на 'b'.

- Входная строка в бинарном виде: «0110 0010 0110 0101 0110 0101 0111 0000 0010 0000 0110 0010 0110 1111 0110 1111 0111 0000 0010 0000 0110 0010 0110 0101 0110 0101 0111 0010 0010 000»
- Закодированная строка: «0011 1110 1011 0001 0010 1010 1100 1111 1000 1001»
- 120/40 = 3 раза!

Дилемма интепретации дерева

- Декодировщик должен иметь в наличии дерево,а также правила его интерпретации.

Jpeg

- Joint Photographics Experts Group — популярный графический формат сжатия изображений (в основном фотографических данных). Использует необратимое сжатие с потерями(возможен режим без потерь). Макс размер 65535x65535 точек

Сжатие с потерями

- Сжатие без потерь: преобразования обратимы.
- Сжатие с потерями а) цветовая информация- постеризация
- Сжатие с потерями б) искажающее сжатие

  главная задача сжатия с потерями- искажение данных несущественны с точки зрения их дальнейшего применения.

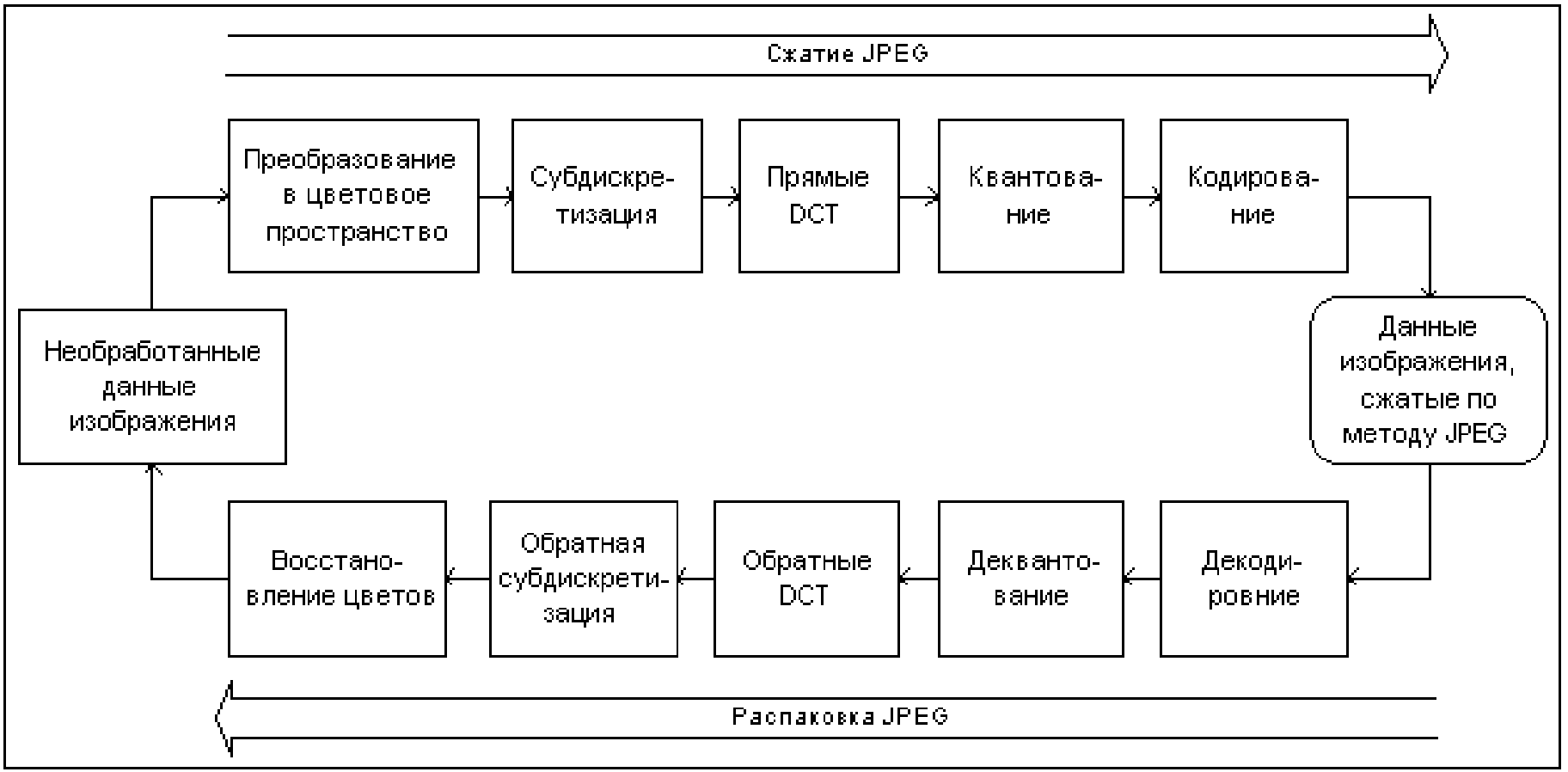
Jpeg- алгоритмика

1. Преобразование в YUV модель

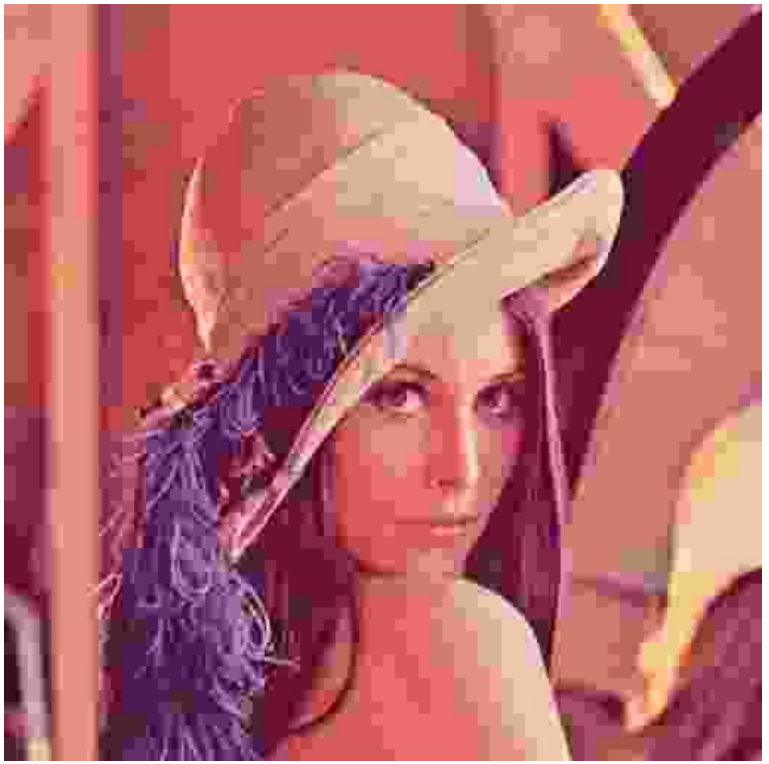
2. Прореживание каналов Cb и Cr 2x2

3. Яркостный компонент — дискретное косинусное преобразование блоков 8x8

4. Полученные коэффициенты квантуются и пакуются кодом Хаффмана. Высокочастотные коэффициенты квантуются более сильно,чем низкочастотные,что приводит к «смазыванию мелких деталей» и появлению «артефактов».

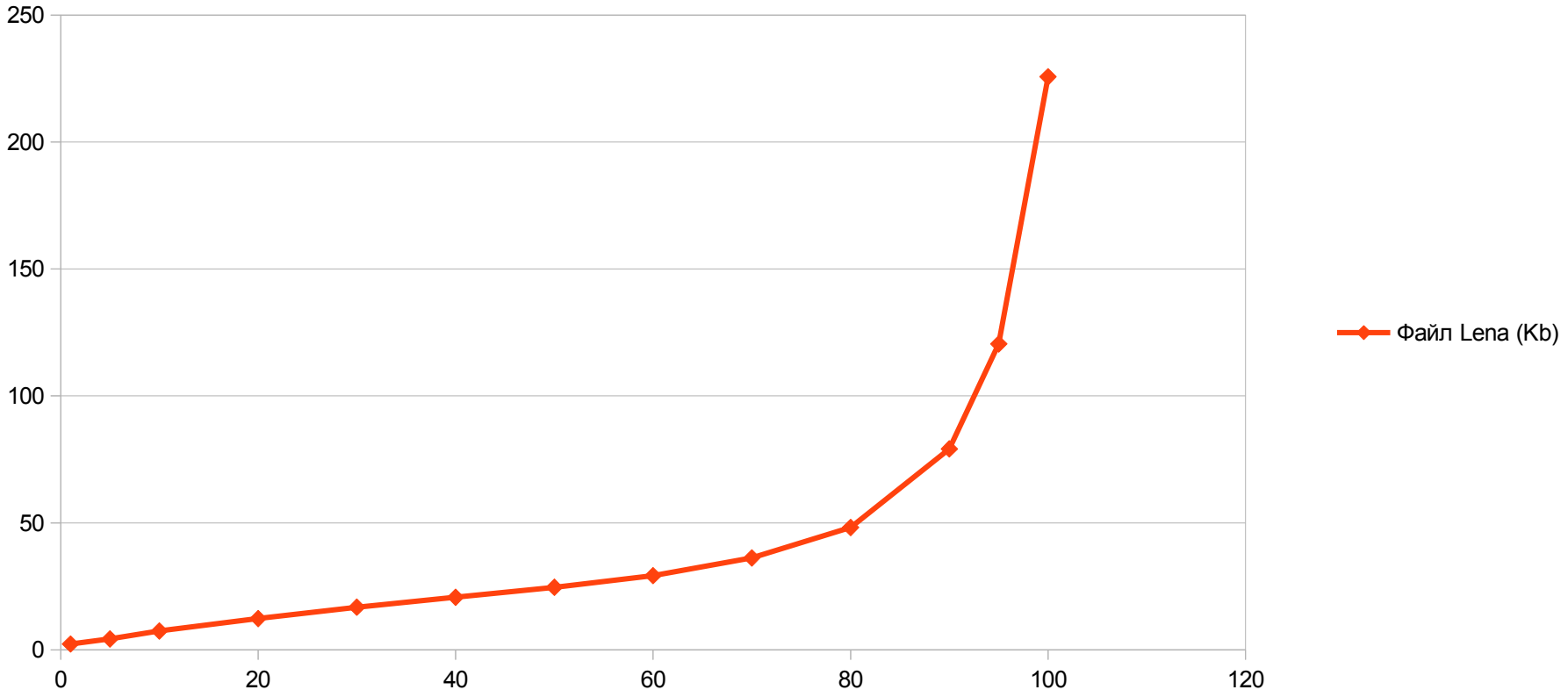

Сжатие JPEG
Преобразование в цветовое пространство
Субдискре-тизация
Прямые DCT
Квантова-ние
Кодирова-ние
Данные изображения, сжатые по методу JPEG
Необработанные данные изображения
Восстано-вление цветов
Обратная субдискрети-зация
Обратные DCT
Деквантo-вание
Декоди-ровние
Распаковка JPEG

Фактор сжатия



Фактор 5- 50

Зависимость размера файла от коэф. качества



Jpeg и постобработка

- Артефакты
- Потеря мелких деталей
- Уменьшение цветового охвата
- Размывание фронтов
  - > для решения задачи цифровой обработки сигналов предпочтительнее применять камеры,спобные передавать данные в RAW(сыром) формате.

Типы устройств(некоторые)

- USB камера

- Ethernet камера

- IEEE1934 FireWire

USB камера

- Для устройств USB 2.0 регламентировано три режима работы:
- Low-speed, 10—1500 Кбит/с (используется для интерактивных устройств: клавиатуры, мыши, джойстики)
- Full-speed, 0,5—12 Мбит/с (аудио-, видеоустройства)
- Hi-speed, 25—480 Мбит/с (видеоустройства, устройства хранения информации)

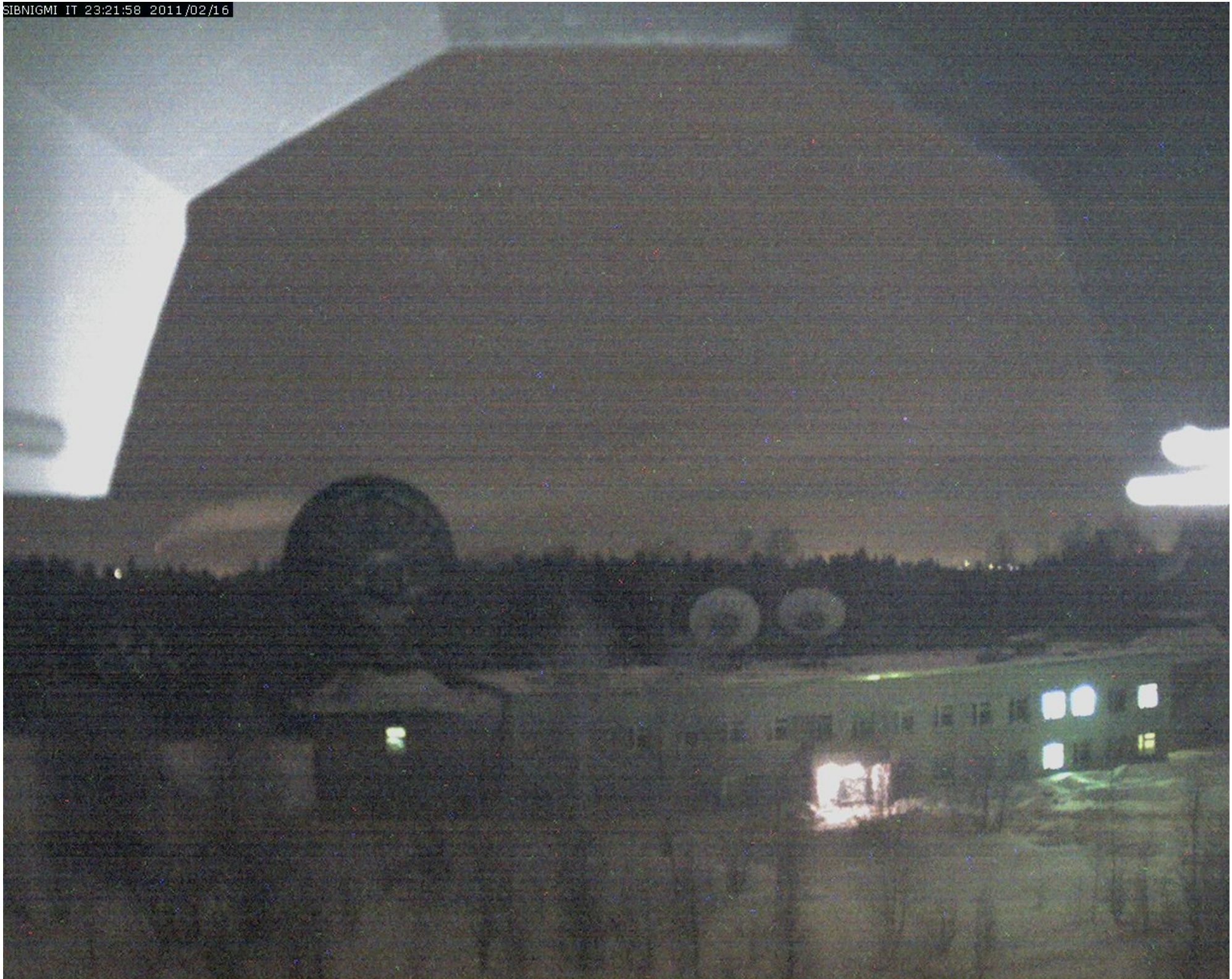
USB web камера

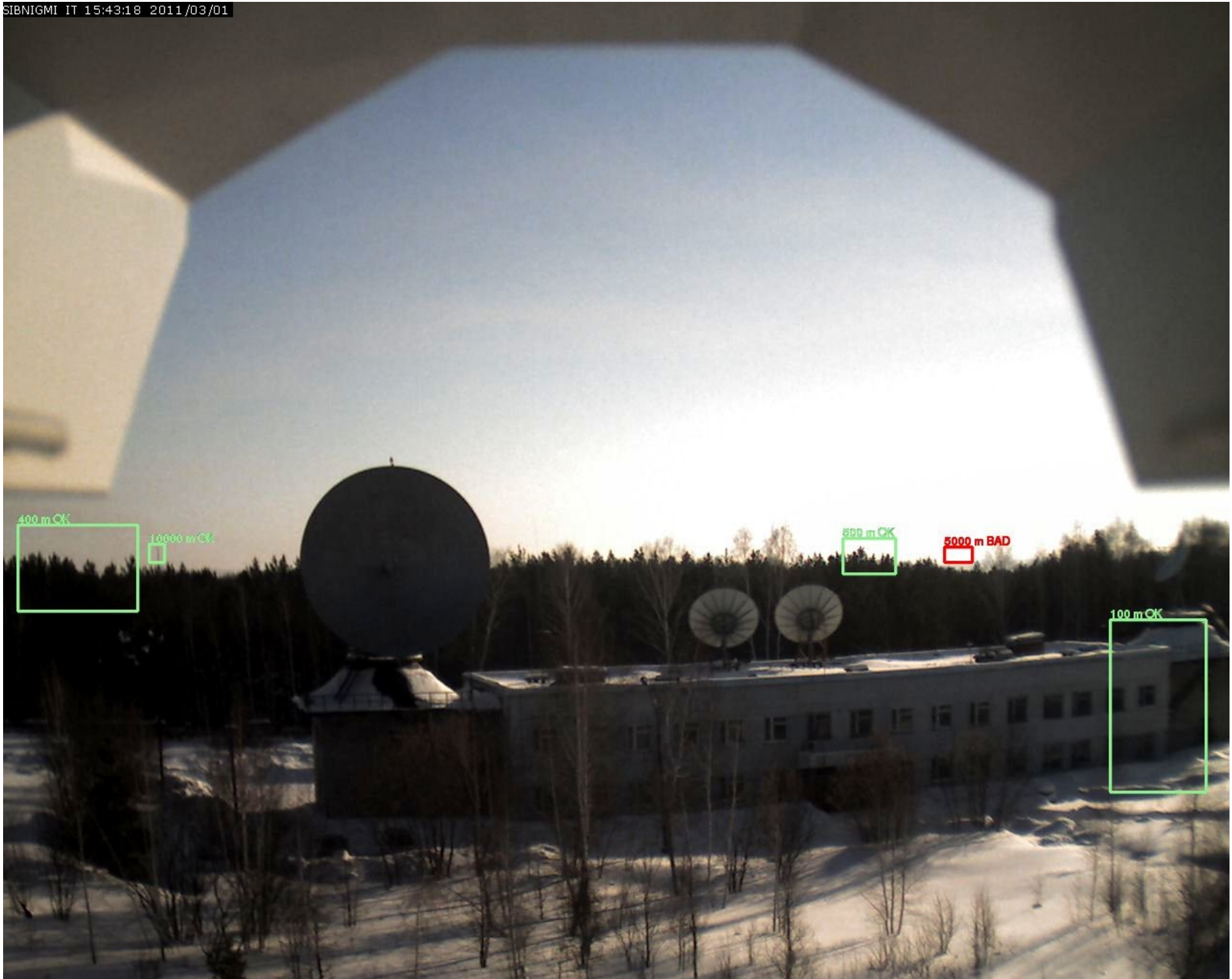
- Реальное разрешение матрицы «скрыто» и зачастую это прос
- Управлять параметрами камеры (диафрагма,ISO) затруднител невозможно)
- Лучшие образцы обеспечивают не более 25 Fps
- Получить данные в Raw формате нельзя (только сжатые с потерей)
- Сигнальный процессор камеры в режима «единичного» кадра может не успеть настроить «диафрагму» и фокус (где это возможно)
- Управлять ROI невозможно
-
- + Дешевое решение из коробки, предоставляющее данные удовлетворительного качества

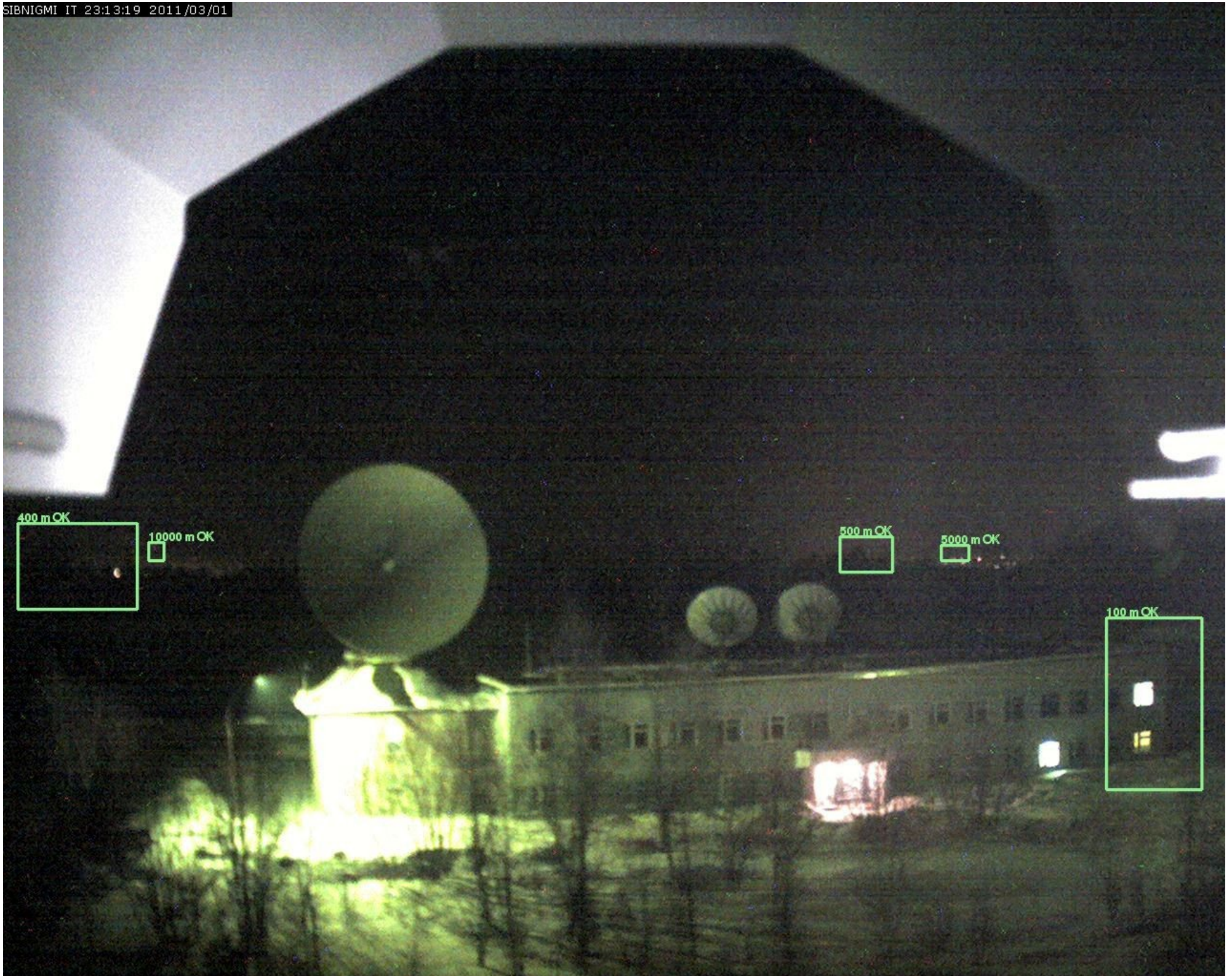
Ethernet камера

- Актуально все что сказано для USB камер, с поправкой на «публичный канал связи»
-
- Интересны «охранные камеры», способные в т.ч. Видеть в ИК диапазоне (например, I
- 

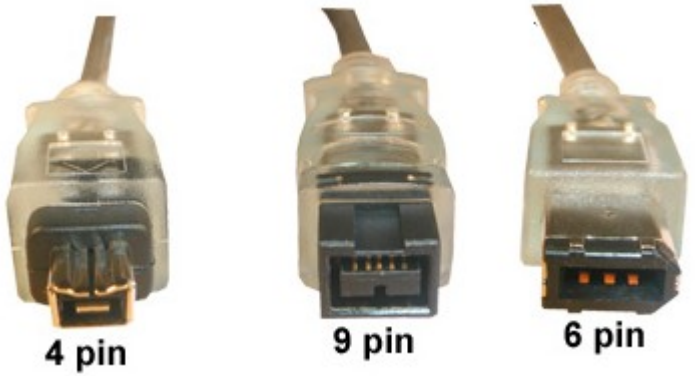

SIBNIGMI IT 10:11:26 2011/02/11


SIBNIGMI IT 23:21:58 2011/02/16

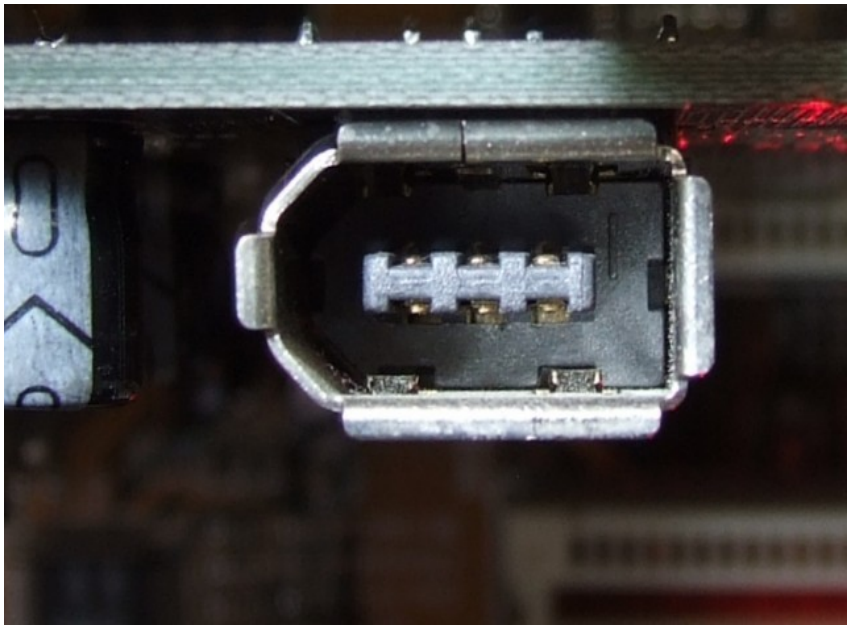

SIBNIGMI IT 15:43:18 2011/03/01
400 m OK
10000 m OK
800 m OK
5000 m BAD
100 m OK

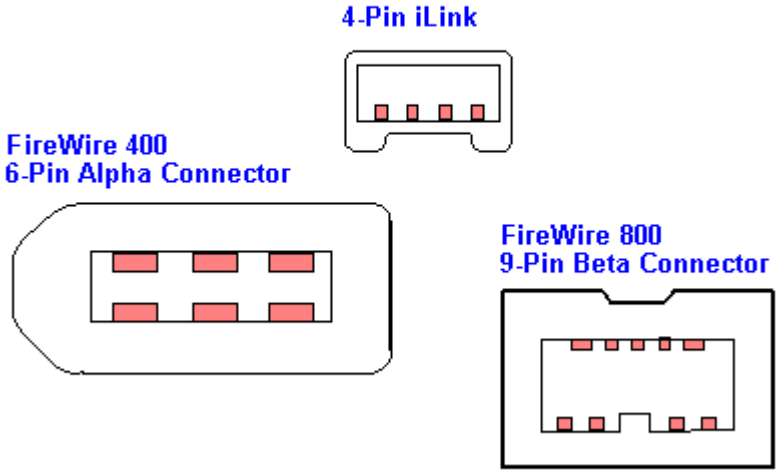

SIBNIGMI IT 23:13:19 2011/03/01
400 m OK
10000 m OK
500 m OK
5000 m OK
100 m OK

FireWire Iee1934



From Computer Desktop Encyclopedia





FireWire

- Различная скорость передачи данных — 100, 200 и 400 Мбит/с в стандарте IEEE 1394/1394a, дополнительно 800 и 1600 Мбит/с в стандарте IEEE 1394b и 3200 Мбит/с в спецификации S3200.
- Наличие питания прямо на шине (маломощные устройства могут обходиться без собственных блоков питания). До полутора ампер и напряжение от 8 до 40 вольт.
- Высокая скорость — возможность обработки мультимедиа-сигнала в реальном времени
- Изохронный трафик - Поток данных, передаваемый с постоянной скоростью, в котором все последовательно передаваемые блоки данных строго взаимно синхронизированы с большой точностью.

  ----> профессиональные скоростные камеры

Marlin Cam

- Пример линейки профессиональных камер.
- До 100 fps
- Возможность установки ROI
- Shutter
- Gain
- Pixel depth

ROI

- Регион интереса — устанавливаем область кадра,которая нас интересует. Только она передается в канал связи

Алгоритм работы

- 1. Работаем с «полным» кадром, ждем появления объекта интереса
- 2. Захватили объект- передали камере регион интереса «с запасом»
- 3. периодически перепозиционируем область ROI

- Для Marlin F146C полный кадр:Up to 17.4 fps
- Выключение препроцессинга и установка ROI позволило поднять скорость до 100 fps!!!

Домашнее задание

- Сформировать набор данных, обеспечивающий наихудшее сжатие в алгоритмах LZW и методе Хаффмана