Media Data Compression

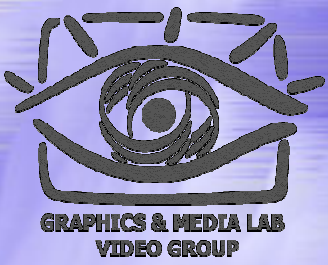
Сжатие без потерь

Дмитрий Ватолин

Московский Государственный Университет
CS MSU Graphics&Media Lab

Version 2.2

Материалы о сжатии



В мае 2002 года на базе нашей лаборатории был создан сервер «Все о сжатии»: **http://www.compression.ru/**. Сейчас сайт содержит более 600Мб информации и является крупнейшим русскоязычным сайтом по сжатию.

На сайте выложена книга Д.Ватолин, М.Смирнов, А.Ратушняк, В.Юкин «Методы сжатия данных», Диалог-МИФИ, 2002. Данный курс дополняет ее в областях сжатия аудио, изображений и 3D-видео.

Цель лекций

GRAPHICS & MEDIA LAB
VIDEO GROUP

Целью данных лекций является рассказ об избранных базовых и новых технологиях, использующихся при сжатии звука, изображений и видео.

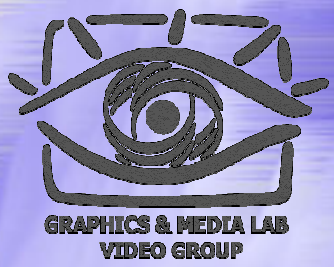
Первыми рассказываются методы сжатия без потерь, базовые для остальных методов.

Структура материала

GRAPHICS & MEDIA LAB
VIDEO GROUP

- Введение
  - Общие понятия сжатия
  - Теорема Шеннона
- Методы сжатия
  - Метод Хаффмана
  - Арифметическое сжатие
  - PPM
  - BWT (MTF)
  - LZ-Huffman

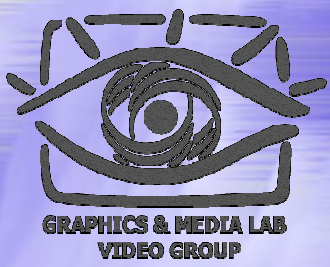
Методы сжатия без потерь



Методы сжатия без потерь разделяют на две категории:

- Методы сжатия источников данных без памяти (т.е. не учитывающих последовательность символов)
- Методы сжатия источников с памятью

Методы сжатия источников без памяти



- **Сжатие по Хаффману**
  Самый известный и распространенный метод. Сдает позиции более мощному арифметическому сжатию.
- **Арифметическое сжатие**
  Наилучший на сегодня метод по степени сжатия. Имеет быструю реализацию, крайне гибок.
- **Сжатие с кодами Райса-Голомба**
  Используется как компромисс между методом Хаффмана и Арифметическим, когда есть ограничения на вычислительную сложность..

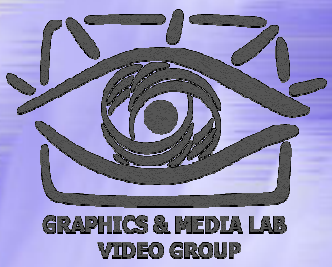
  (также известны нумерующие кодирование, разделение мантисс и экспонент, коды Элиаса, Фибоначчи и др.)

Методы сжатия источников с памятью

GRAPHICS & MEDIA LAB
VIDEO GROUP

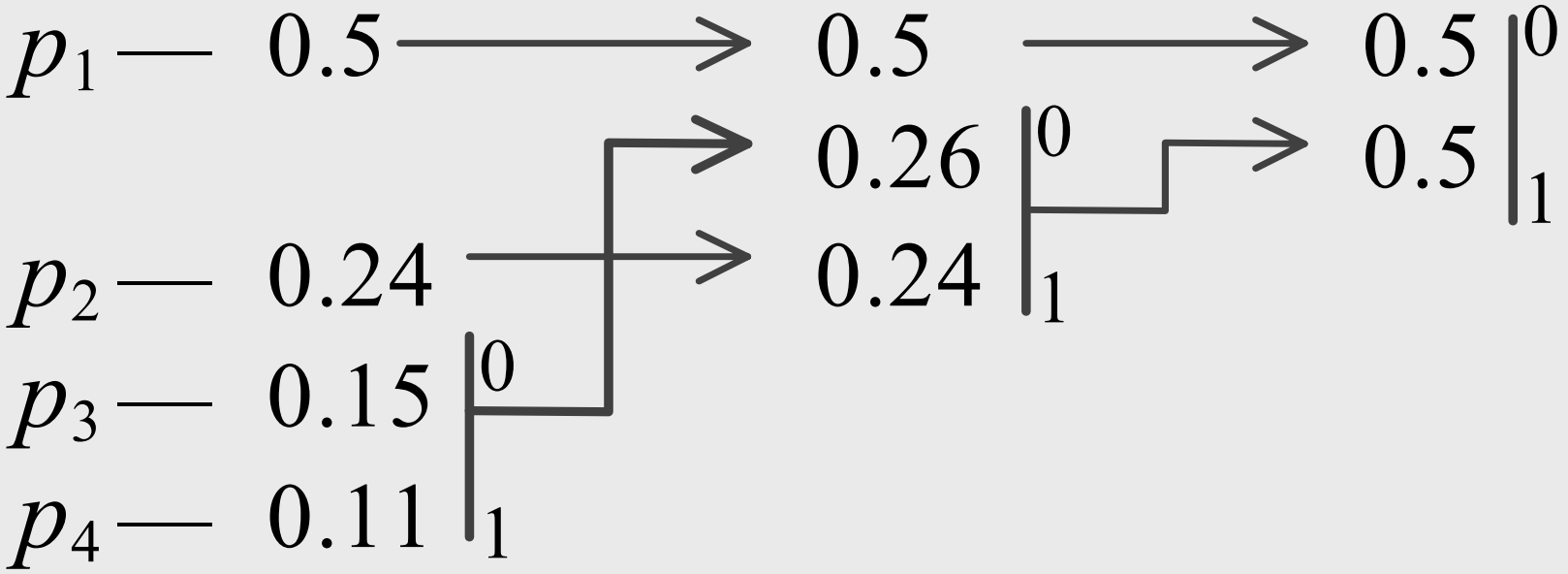
- **Словарные методы сжатия**
  (LZ, LZW. Давний универсальный метод (ZIP), используется для сжатия в GIF & PNG)
- **Методы контекстного моделирования**
  (PPM. Новый универсальный метод. Позволяет добиться максимальных результатов.)
- **Сжатие с использованием преобразования Барроуза-Уилера и других сортирующих преобразований**
  (BWT, ST. Используется в основном для текста.)

Алгоритм Хаффмана



Использует только частоту появления одинаковых байт в изображении. Сопоставляет символам входного потока, которые встречаются большее число раз, цепочку бит меньшей длины. И, напротив, встречающимся редко — цепочку большей длины. Для сбора статистики требует двух проходов по изображению.

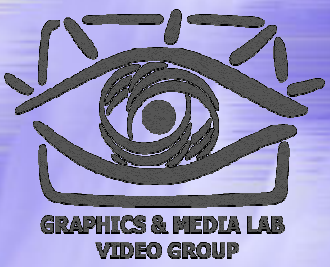
Алгоритм Хаффмана-2



Алгоритм Хаффмана-3

- **Коэффициенты компрессии:** 8, 1,5, 1 (Лучший, средний, худший коэффициенты).
- **Использование:** Практически не применяется в чистом виде. Обычно используется как один из этапов компрессии в более сложных схемах.
- **Симметричность:** 2 (за счет того, что требует двух проходов по массиву сжимаемых данных).
- **Характерные особенности:** Единственный алгоритм, который не увеличивает размера исходных данных в худшем случае (если не считать необходимости хранить таблицу перекодировки вместе с файлом).

Теорема Шеннона



Теорема о кодировании источника: Элемент s_i, вероятность появления которого равняется $p(s_i)$, выгоднее всего представлять $-\log_2 p(s_i)$ битами. Если при кодировании размер кодов всегда в точности получается равным $-\log_2 p(s_i)$ битам, то в этом случае длина закодированной последовательности будет минимальной для всех возможных способов кодирования.

Энтропия источника

Если распределение вероятностей $F = \{p(s_i)\}$ неизменно, и вероятности появления элементов независимы, то мы можем найти **среднюю длину кодов** как среднее взвешенное:

$$H = -\sum_i p(s_i) \cdot \log_2 p(s_i)$$

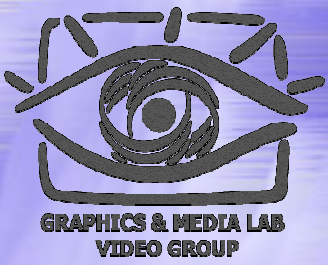
Это значение также называется ***энтропией распределения вероятностей F*** или ***энтропией источника в заданный момент времени***.

Структура материала

GRAPHICS & MEDIA LAB
VIDEO GROUP

- Введение
  - Общие понятия сжатия
  - Теорема Шеннона
- Методы сжатия
  - Метод Хаффмана
  - **Арифметическое сжатие**
  - PPM
  - BWT (MTF)
  - LZ-Huffman

Арифметическое сжатие

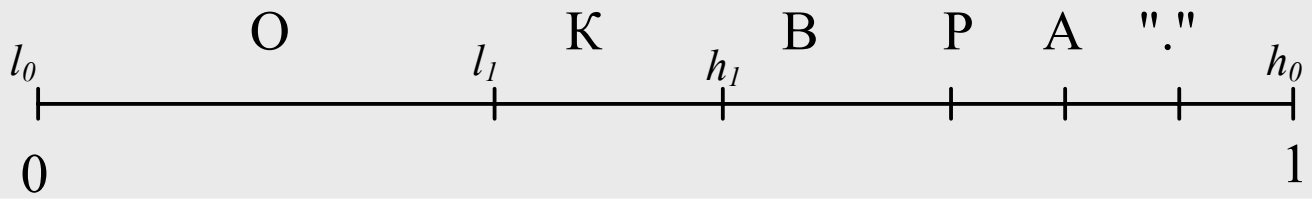


Основная идея: Мы представляем кодируемый текст в виде длинной дроби. Для этого берется интервал [0, 1) (0 — включается, 1 — нет), который разбивается на подынтервалы с длинами, пропорциональными вероятностям появления символов в потоке.

АС: Пример

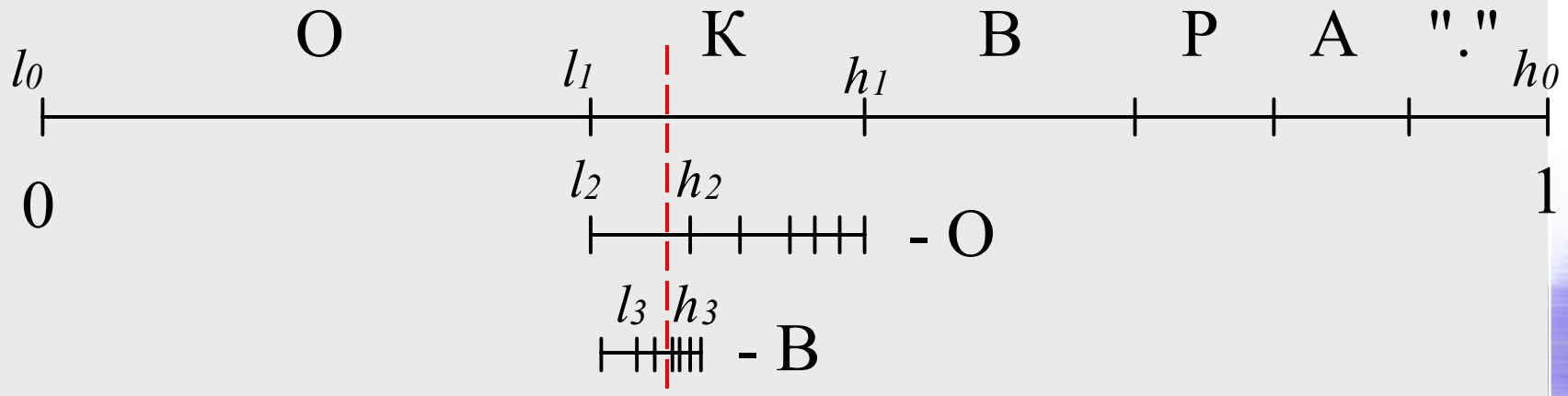
Пусть мы сжимаем текст "КОВ.КОРОВА"

Символ	Вероятность	Интервал
О	0.3	[0.0; 0.3)
К	0.2	[0.3; 0.5)
В	0.2	[0.5; 0.7)
Р	0.1	[0.7; 0.8)
А	0.1	[0.8; 0.9)
"."	0.1	[0.9; 1.0)

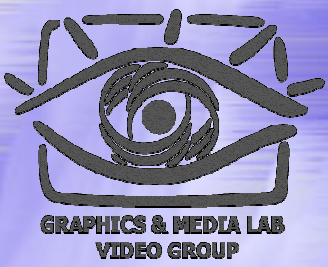


АС: Визуальное представление

Графически соответствующую процедуру можно представить так:



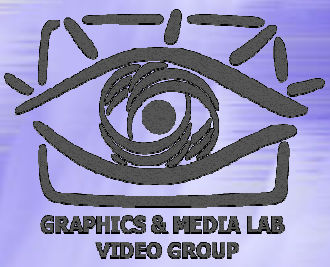
АС: Пример



Берем исходный интервал и кодируем текст:

Изначально интервал			[0, 1).
Символ "К"	[0.3; 0.5)	получаем	[0.3; 0.5).
Символ "О"	[0.0; 0.3)	получаем	[0.3; 0.36).
Символ "В"	[0.5; 0.7)	получаем	[0.33; 0.342).
Символ "."	[0.9; 1.0)	получаем	[0,3408; 0.342).

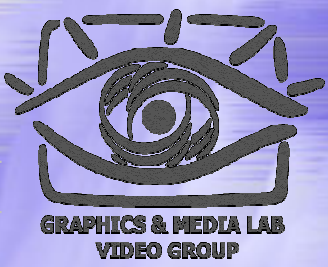
AC: Процедура сжатия



Если обозначить интервал символа c, как $[a[c]; b[c])$, а кодируемый интервал для i-го символа потока как $[l_i, h_i)$. То алгоритм компрессии запишется как:

```
l0=0; h0=1; i=0;
while(not DataFile.EOF()){
  c = DataFile.ReadSymbol(); i++;
  li = li-1 + a[c]·(hi-1 - li-1);
  hi = li-1 + b[c]·(hi-1 - li-1);
};
```

АС: Процедура распаковки

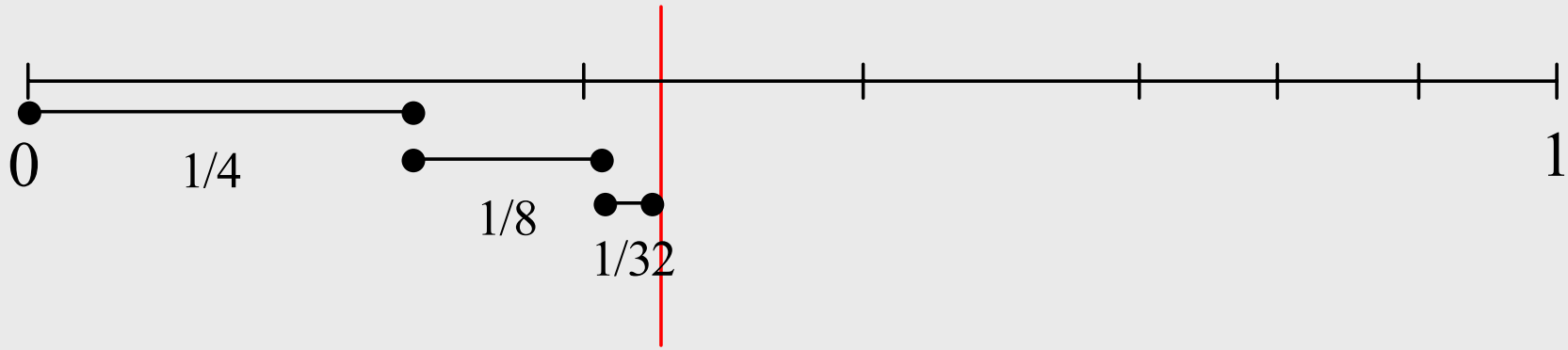


Алгоритм декомпрессии выглядит так:

```
l_0=0; h_0=1; value=File.Code();
for(i=0; i<File.DataLength(); i++){
  for(all symbols c_j){
     l_i = l_{i-1} + a[c_j]·(h_{i-1} - l_{i-1});
     h_i = l_{i-1} + b[c_j]·(h_{i-1} - l_{i-1});
     if (l_i <= value < h_i) break;
  };
  DataFile.WriteSymbol(c_j);
};
```

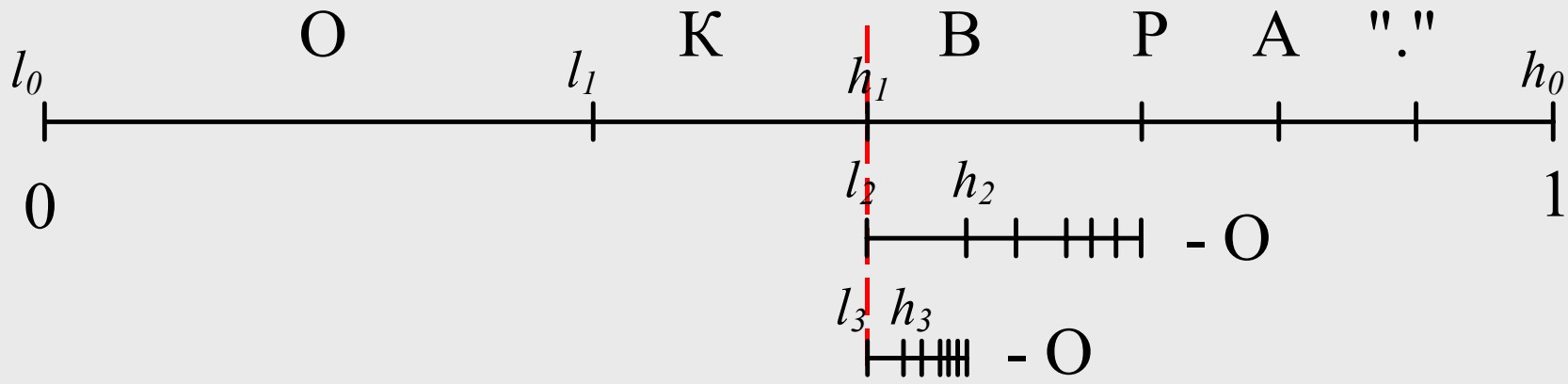
АС: Двоичные дроби

Заметим, что мы можем приближать получающуюся дробь с помощью двоичной дроби

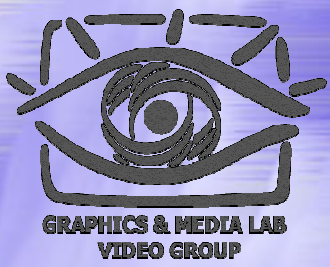


АС: Бесконечное сжатие

Пример: один бит "1" (двоичное число "0.1") для наших интервалов однозначно задает цепочку "ВОООООООООО…" **сколь угодно большой длины**.



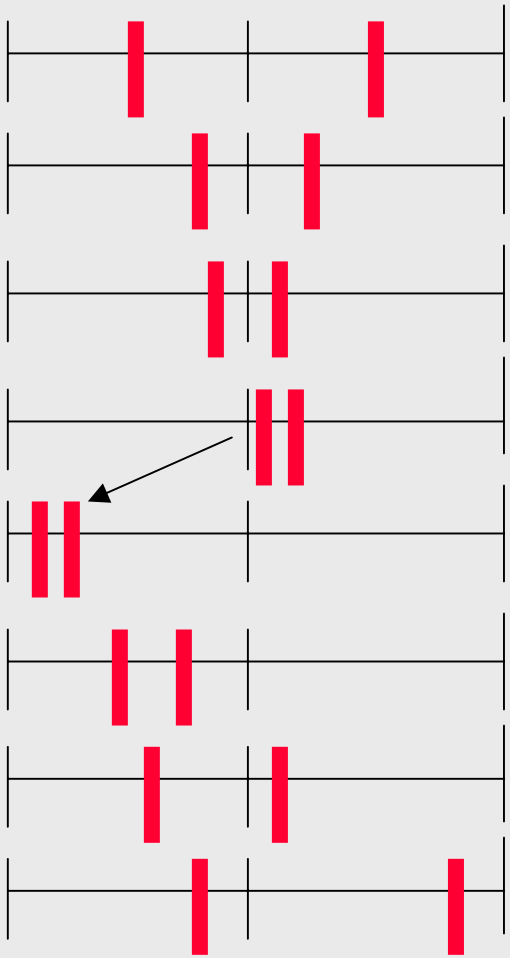
AC: Целочисленные вероятности



Перейдем к целочисленным коэффициентам:

J	Символ (c_j)	Вероятность	$b[c_j]$
0	—	—	0
1	О	3	3
2	К	2	5
3	В	2	7
4	Р	1	8
5	А	1	9
6	"."	1	10

AC: Пример нормализации



В выходной поток

‘1’

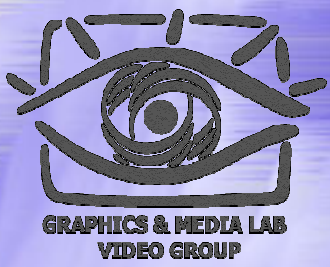
‘0’

‘0’

‘0’

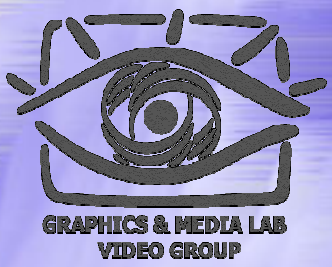
Движение подынтервалов при реальном сжатии

AC: Реальный пример процедуры сжатия



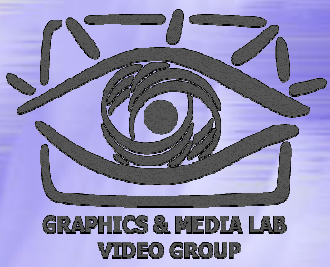

```
l_0=0; h_0=65535; i=0; delitel= b[c_last]; // =10
First_qtr = (h_0+1)/4; Half = First_qtr*2; // = 16384 = 32768
Third_qtr = First_qtr*3; bits_to_follow =0; // = 49152, Сколько бит сбрасывать

while(not DataFile.EOF()) {
    c = DataFile.ReadSymbol();  // Читаем символ
    j = IndexForSymbol(c); i++  // Находим его индекс
    l_i = l_i-1 + b[j-1]*(h_i-1 - l_i-1 + 1)/delitel;
    h_i = l_i-1 + b[j  ]*(h_i-1 - l_i-1 + 1)/delitel - 1;
    for(;;) {                   // Обрабатываем варианты
          if(h_i < Half)        // переполнения
                    bits_plus_follow(0);
          else if(l_i >= Half) {
                    bits_plus_follow(1);
                    l_i-= Half; h_i-= Half;
          }
          else if((h_i < First_qtr)&&(l_i >= Third_qtr)){
                    bits_to_follow++;
                    l_i-= First_qtr; h_i-= First_qtr;
          } else break;
          l_i+=l_i; h_i+= h_i+1;
    }
}
```

АС: Реальный пример процедуры сжатия (2)




```
 // Процедура сброса найденных бит в файл
void bits_plus_follow (int bit)
{
   CompressedFile.WriteBit(bit);
   for(; bits_to_follow > 0; bits_to_follow--)
       CompressedFile.WriteBit(!bit);
}
```

АС: Работа целочисленного алгоритма



Пример сжатия цепочки:

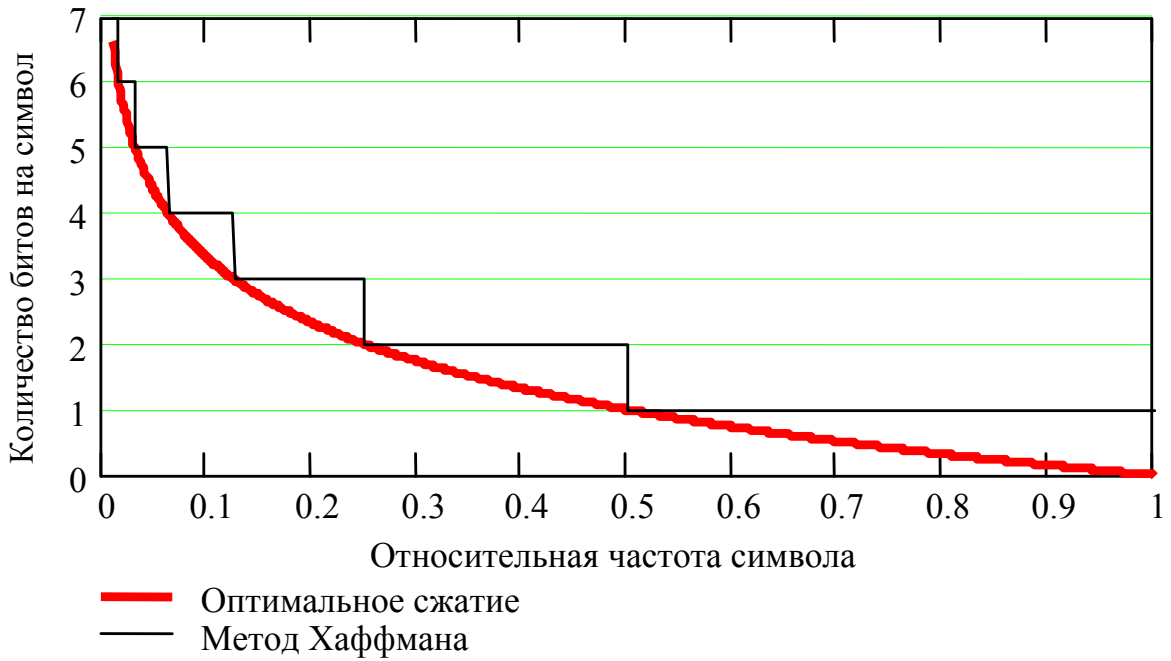
i	Символ (c_j)	h_i	l_i	Нормализ: h_i	Нормализ: l_i	Вывод
0		65535	0			
1	К	32767	19660	65535	13104	01
2	О	28832	13104	57665	26208	010
3	В	48227	41937	58143	7816	010101
4	.	58143	53111	35967	15836	01010111
5	К	25901	21875	38071	21964	0101011101
6	О	26795	21964	41647	22320	010101110101

АС: Характеристики

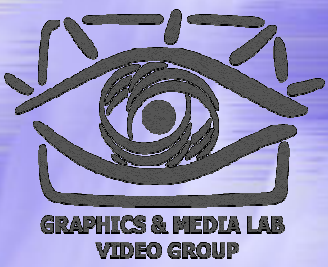
Характеристики арифметического сжатия:

- Позволяет сжимать несколько сильнее, чем алгоритм Хаффмана
- Работает медленнее, чем алгоритм Хаффмана
- Допускает как статическую, так и динамическую (адаптивную) реализацию

AC: Сравнение с алгоритмом Хаффмана



AC: Пример

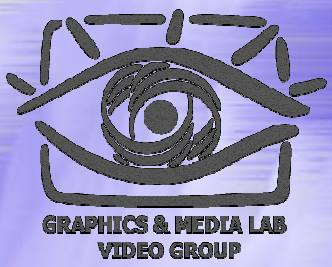


Пусть есть два символа *a* и *b* с вероятностями 253/256 и 3/256

Для арифметического сжатия мы потратим на цепочку из 256 байт $-\log_2(253/256)\cdot 253 - \log_2(3/256)\cdot 3 = 23.546$, т.е. **24 бита**.

При кодировании по Хаффману мы закодируем *a* и *b* как 0 и 1, и потратим $1\cdot 253+1\cdot 3=$**256 битов**, т.е. **в 10 раз больше**

Повышение степени сжатия



Методы повышения степени сжатия:

- Применение динамических таблиц
- Изменение агрессивности динамической подстройки
- Инициализация таблиц (несколько таблиц)
- Использование переключения между таблицами
- Увеличение точности вычислений (в int & double)
- Использование PPM

Структура материала

GRAPHICS & MEDIA LAB
VIDEO GROUP

- Введение
  - Общие понятия сжатия
  - Теорема Шеннона
- Методы сжатия
  - Метод Хаффмана
  - Арифметическое сжатие
  - **PPM**
  - BWT (MTF)
  - LZ-Huffman

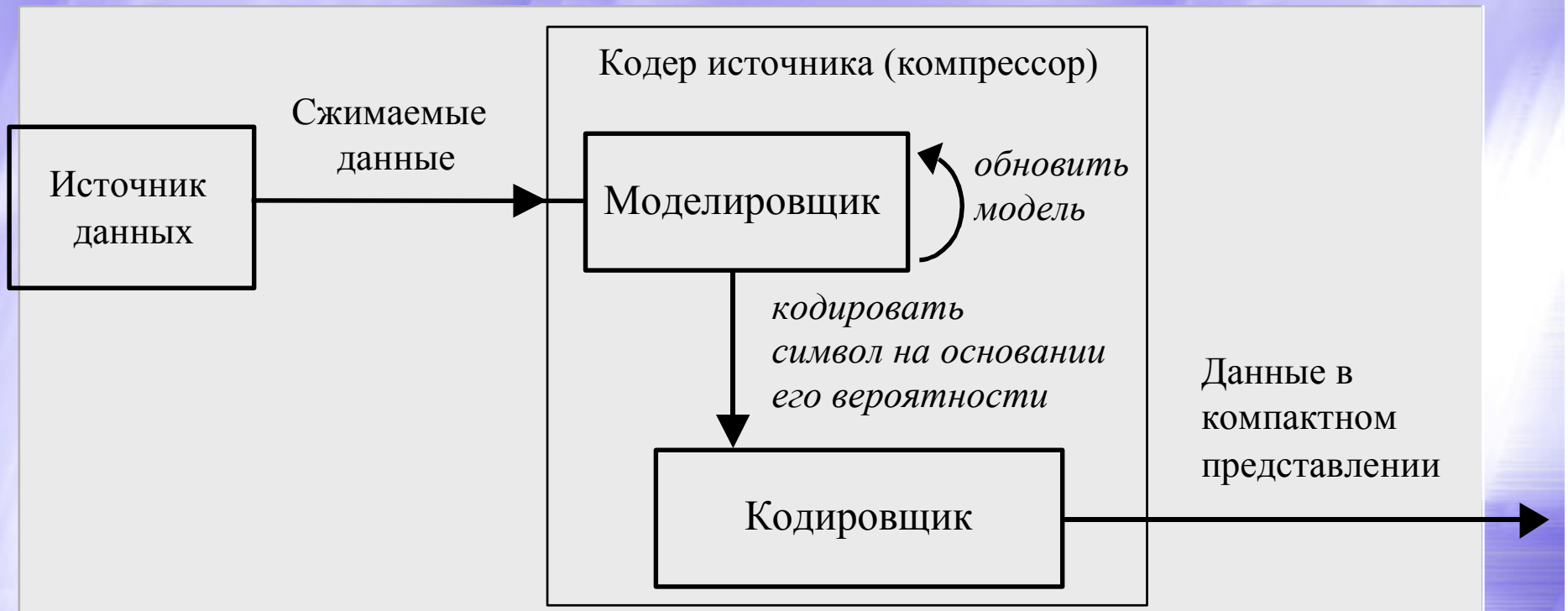
PPM: Идея

Классический PPM (prediction by partial matching) - это метод контекстно-зависимого моделирования ограниченного порядка, позволяющий оценить вероятность символа в зависимости от предыдущих символов.

Строку символов, непосредственно предшествующую текущему символу, будем называть ***контекстом***.

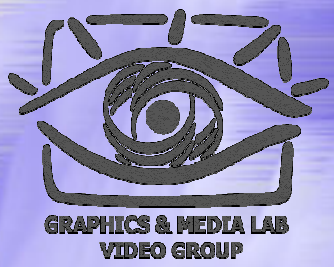
Если для оценки вероятности используется контекст длины N, то мы имеем дело с ***контекстно-ограниченной моделью степени N*** или порядка N.

PPM: Общая схема алгоритма



Важно, что каждый новый символ кодируется на оценке его вероятности

PPM: Пример модели 0

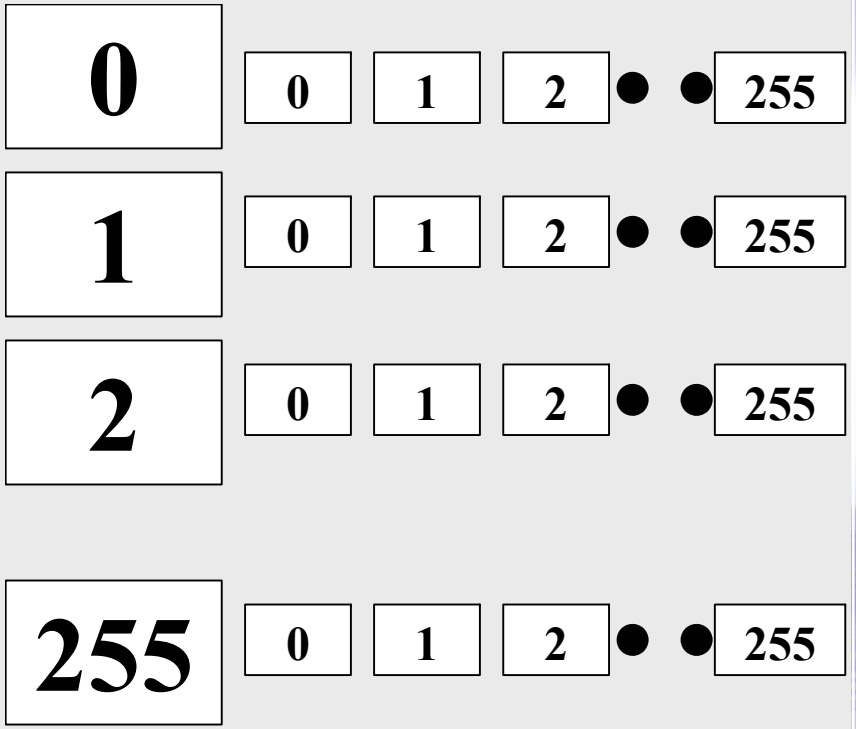


Простой пример – модель порядка 0: тогда вероятность следующего символа будет зависеть от того, как часто он встречался ранее.

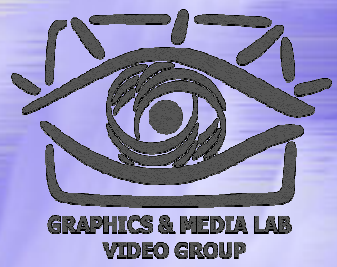
0
1
2
255

PPM: Пример модели 1

Простой пример – модель порядка 1: тогда вероятность следующего символа будет зависеть от предыдущего символа.

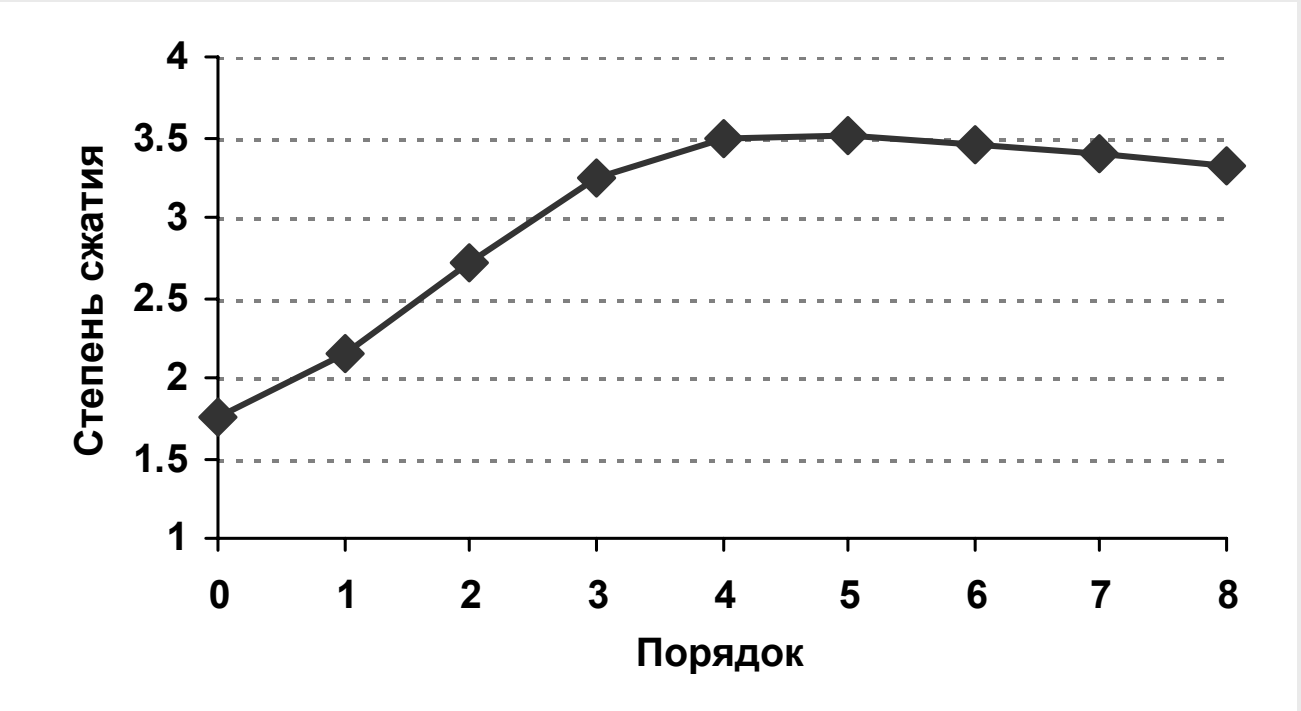


PPM: варианты моделирования



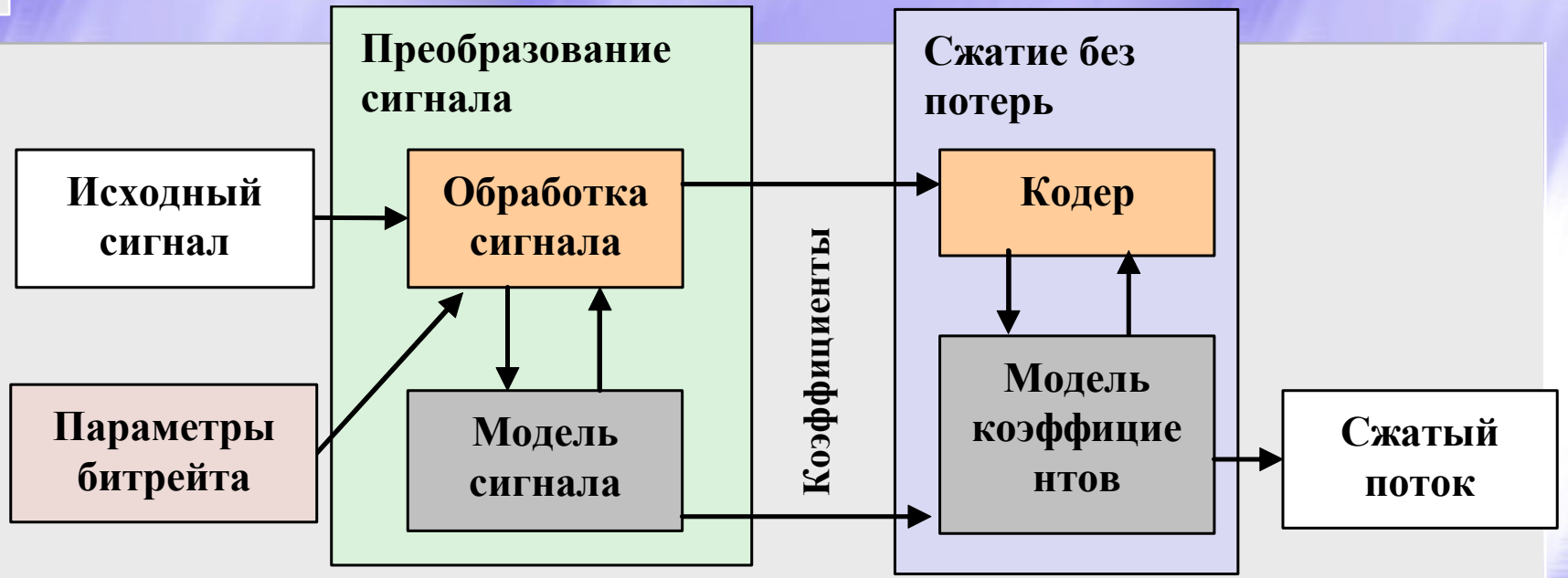
- Статическое
  Используется фиксированная модель
- Полуадаптивное
  Модель сохраняется в файле
- Адаптивное (динамическое)
  Модель изменяется в процессе сжатия и распаковки
- Блочно-адаптивное
  Модель меняется сильно между блоками разных данных

PPM: Выбор сложности модели



Зависимости степени сжатия от длины модели для текстовых данных

PPM: Принципы сжатия сигналов



В модели сигнала - используются знания о важности частей сигнала для человека

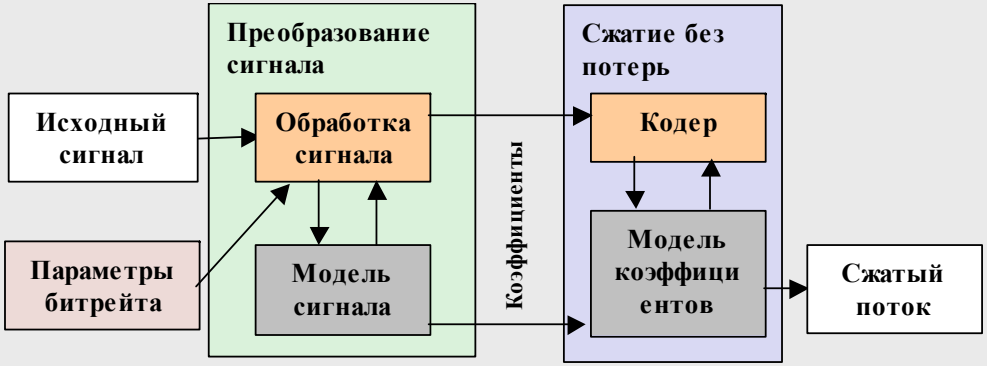
В модели коэффициентов используются знания об избыточности коэффициентов.

PPM: Сжатие изображений

Используется преобразование цветовых пространств и т.д.

Модели сигнала:

- DCT
- Wavelets
- Fractals (Аффинное преобразование)

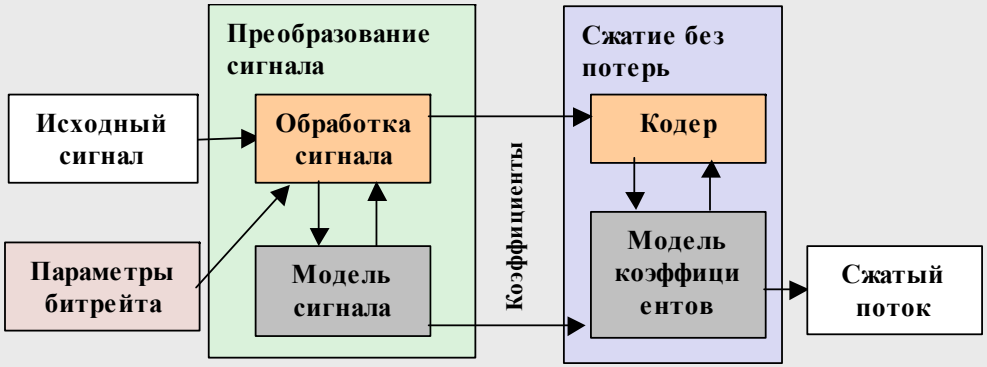


PPM: Сжатие видео

- Используется преобразование цветовых пространств (избыточность по цвету).
- Используется компенсация движения (избыточность по времени).

Модели сигнала:

- DCT
- Wavelet
- Object-oriented

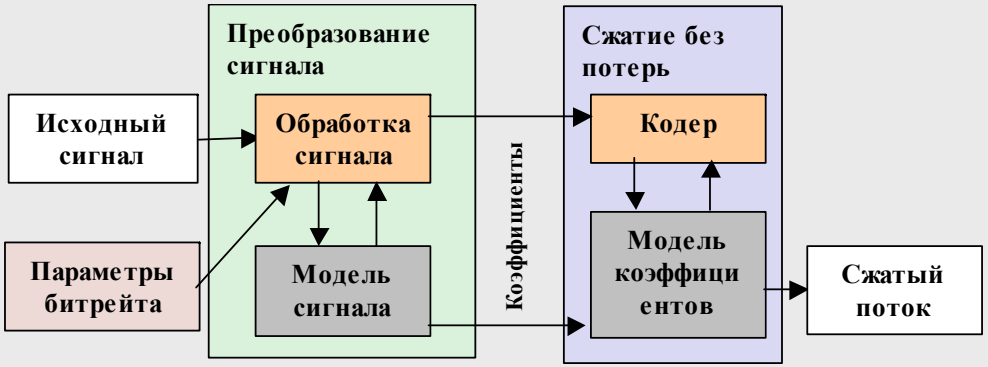


PPM: Сжатие звука

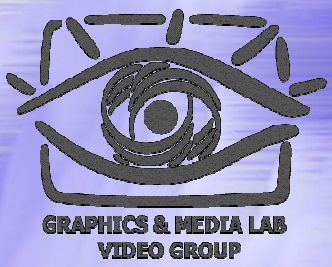
- Используется маскирование по частоте (избыточность по частоте).
- Используется маскирование по времени.
- Используется избыточность стерео-сигнала.

Модели сигнала:

- MDCT
- DCT
- FFT
- Wavelets



Задача: общая постановка

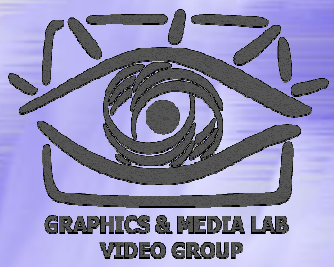


- Программа умеет получать на вход файл и по опции «c» - сжимать его, по опции «d» распаковывать его.
- Задается метод сжатия – арифметический (обязателен) и PMM.
- Язык реализации – консольное приложение на C или C++

Пример:

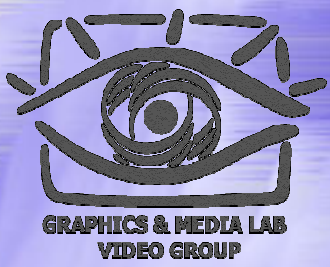
```
compress c in_file.txt  out_file.cmp ppm
compress d out_file.txt out_file.txt
```

Задача: Требования



- Арифметическое сжатие – только классический алгоритм (методы его оптимизации разбирались)
- За использование чужих текстов или совместное написание – дисквалификация.
- Оцениваться будет степень сжатия файлов, отдаваемых на вход программы.
- **Распакованный файл должен совпадать с паковавшимся!!!**

Задача: Улучшение результата



Методы повышения степени сжатия:

- Применение динамических таблиц
- Изменение агрессивности динамической подстройки
- Инициализация таблиц (несколько таблиц)
- Использование переключения между таблицами
- Увеличение точности вычислений (в int & double)

Задача: Сроки

GRAPHICS & MEDIA LAB
VIDEO GROUP

- Срок начала задания – 15 октября
- Срок сдачи задания – 05 ноября
- Сдаются:
  - Исходный текст в виде компилируемого проекта
  - Пояснения (read_me) с указанием фамилии, группы и номера зачетной книжки
  - Скомпилированная программа и пример
- Готовое задание высылается по адресу **c-course-a1@compression.ru**

Структура материала

GRAPHICS & MEDIA LAB
VIDEO GROUP

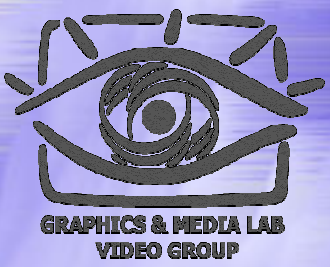
- Введение
  - Общие понятия сжатия
  - Теорема Шеннона
- Методы сжатия
  - Метод Хаффмана
  - Арифметическое сжатие
  - PPM
  - **BWT (MTF)**
  - LZ-Huffman

BWT / Идея

BWT (Burrows-Wheeler Transform) – преобразование Бароуза-Уилера – предназначено для того, чтобы сделать сжатие потока данных более эффективным. Алгоритм опубликован в 1994 (разработан – в 1983).

Мы переставляем символы выходного потока таким образом, что применяемый далее алгоритм становится более эффективен.

BWT / Шаг 1



Пусть мы сжимаем строку символов «**абракадабра**».

Подготовим все ее циклические перестановки.

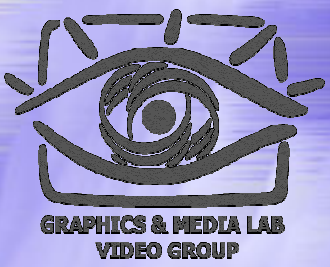
абракадабра
бракадабраа
ракадабрааб
акадабраабр
кадабраабра
адабраабрак
дабраабрака
абраабракад
браабракада
раабракадаб
аабракадабр

BWT / Шаг 2

Пометим в получившейся матрице исходную строку и отсортируем все строки в соответствии с лексикографичес-ким порядком символов.

0	аабракадабр
1	абраабракад
2	**абракадабра** – исх. строка
3	адабраабрак
4	акадабраабр
5	браабракада
6	бракадабраа
7	дабраабрака
8	кадабраабра
9	раабракадаб
10	ракадабрааб

BWT / Шаг 3



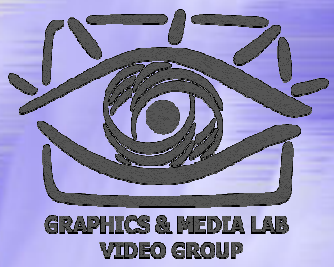
Выписываем символы последнего столбца и запоминаем номер исходной строки среди отсортированных.

Получаем результат преобразования BWT: **«рдакрааааабб», 2**

Длина результата и состав символов – как в исходной цепочке.

аабракадаб**р**
абраабракад
абракадабра - 2
адабраабра**к**
акадабрааб**р**
браабракада
бракадабра**а**
дабраабрака
кадабраабр**а**
раабракада**б**
ракадабраа**б**

BWT / Суть

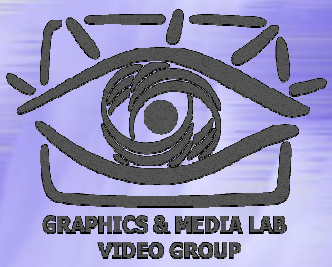


«Фокус» BWT в том, что полученной цепочки **«рдакрааааабб»** и числа 2 достаточно для воссоздания исходной цепочки.

Зачем это нужно? Если мы преобразуем таким образом достаточно длинный текст, со словами ***the, The, then, when, that***, то мы получим на выходе цепочку в которой будет столько ***t*** подряд, сколько слов ***the*** в исходной цепочке, потом будет идти столько ***T***, сколько ***The*** и т.д. Происходит сортировка по «частоте сочетаний»

```
.............
he      ... t
he      ... t
he      ... t
he      ... t
he      ...T
he      ... t
he      ... t
hen     ... t
hen     ...w
hen     ... t
hen     ... t
.............
```

Обратное BWT / Шаг 1



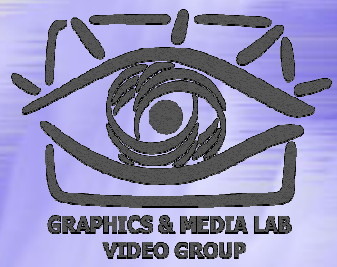
Итак! Мы получили на вход In={**рдакрааааабб}, 2**

Отсортируем полученную цепочку символов.

Нам известно, что строки матрицы были отсортированы по порядку, начиная с первого символа. Поэтому в результате такой сортировки мы получили первый столбец исходной матрицы.

0	**а**
1	**а**
2	**а**
3	**а**
4	**а**
5	**б**
6	**б**
7	**д**
8	**к**
9	**р**
10	**р**

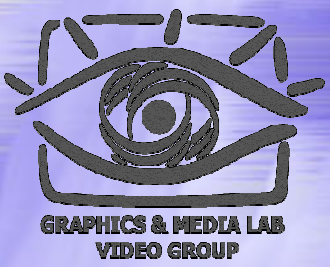
Обратное BWT / Шаг 2



Поскольку последний столбец по условию задачи нам известен, добавим его в полученную матрицу.

0	**а.....р**
1	**а.....д**
2	**а.....а**
3	**а.....к**
4	**а.....р**
5	**б.....а**
6	**б.....а**
7	**д.....а**
8	**к.....а**
9	**р.....б**
10	**р.....б**

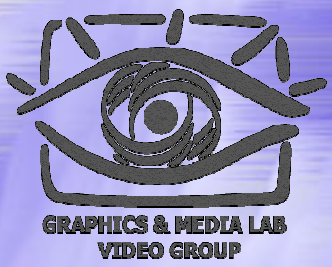
Обратное BWT / Шаг 3



Строки матрицы были получены в результате циклического сдвига исходной строки. То есть, символы последнего и первого столбцов образуют друг с другом пары. И нам ничто не может помешать отсортировать эти пары, поскольку обязательно существуют такие строки в матрице, которые начинаются с этих пар. Заодно допишем в матрицу и последний столбец (**рдакрааааббб**).

0 аа.....р
1 аб.....д
2 аб.....а
3 ад.....к
4 ак.....р
5 бр.....а
6 бр.....а
7 да.....а
8 ка.....а
9 ра.....б
10 ра.....б

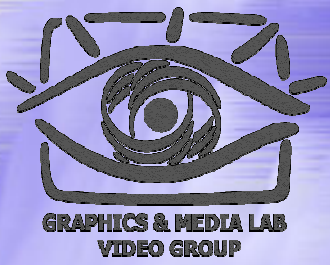
Обратное BWT / Идея



Заметим, что шаг 3 можно повторить еще раз, отсортировав уже тройки символов.

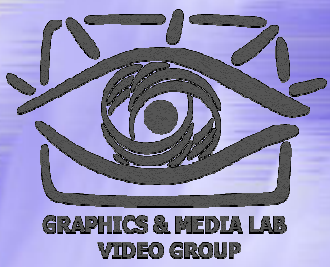
Повторяем этот шаг столько раз, сколько необходимо для восстановления всей таблицы, а потом берем из нее строку с номером 2 в качестве исходной.

Обратное BWT полностью



0	ааб.....р	аабр.....р		аабракада.р	аабракадабр
1	абр.....д	абра.....д		абраабрак.д	абраабракад
2	абр.....а	абра.....а		абракадаб.а	**абракадабра**
3	ада.....к	адаб.....к		адабраабр.к	адабраабрак
4	ака.....р	акад.....р		акадабраа.р	акадабраабр
5	бра.....а	браа.....а	...	браабрака.а	браабракада
6	бра.....а	брак.....а		бракадабр.а	бракадабраа
7	даб.....а	дабр.....а		дабраабра.а	дабраабрака
8	кад.....а	када.....а		кадабрааб.а	кадабраабра
9	раа.....б	рааб.....б		раабракад.б	раабракадаб
10	рак.....б	рака.....б		ракадабра.б	ракадабрааб

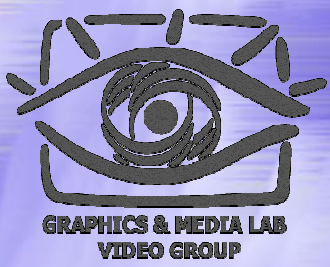
Обратное BWT
Быстрый вариант (1)



Запишем порядок строк после сортировки и перед сортировкой.

номер строки		номер новой строки	переносим последний столбец
2	а.....а	0	аа.....р
5	б.....а	1	аб.....д
6	б.....а	2	аб.....а
7	д.....а	3	ад.....к
8	к.....а	4	ак.....р
9	р.....б	5	бр.....а
10	р.....б	6	бр.....а
1	а.....д	7	да.....а
3	а.....к	8	ка.....а
0	а.....р	9	ра.....б
4	а.....р	10	ра.....б

Обратное BWT
Быстрый вариант (2)



Полученный вектор T = { 2, 5, 6, 7, 8, 9, 10, 1, 3, 0, 4 }, содержит номера позиций символов, упорядоченных в соответствии с положением в алфавите, в строке, которую нам надо декодировать.

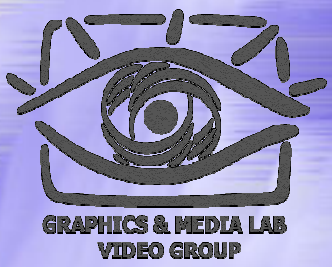
Начинаем декодирование со второго элемента. T[2] = 6, T[6] = 10, T[10] = 4, T[4] = 8…

Получаем цепочку: 6, 10, 4, 8, 3, 7, 1, 5, 9, 0, 2

А теперь, выписываем соответствующие символы из исходной цепочки (In[6], In[10]…). Получаем **абракадабра**

2	**0**	**р**
5	**1**	**д**
6	**2**	**а**
7	**3**	**к**
8	**4**	**р**
9	**5**	**а**
10	**6**	**а**
1	**7**	**а**
3	**8**	**а**
0	**9**	**б**
4	**10**	**б**

BWT – Характеристики



Характеристики BWT:

- Работает сравнительно медленно
- Требует достаточно много памяти
- Позволяющее значительно поднять степень сжатия, в особенности на текстовых данных.

Метод MTF

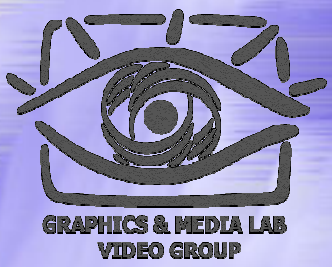
Метод MTF (Move To Front) – русское название «метод стопки книг»

Идея крайне проста:

- Мы получаем из потока новый символ (название книги),
- Помещаем в выходной поток ее номер в стопке
- Кладем книгу в начало стопки

.....	
he	... t
he	... t
he	... t
he	... t
he	...T
he	... t
he	... t
hen	... t
hen	...w
hen	... t
hen	... t
.....	

Метод MTF / Псевдокод



N – число символов в алфавите.

M[N] – упорядоченный список символов. M[0] соответствует верхней книге стопки, M[N-1] — нижней.

x – очередной символ

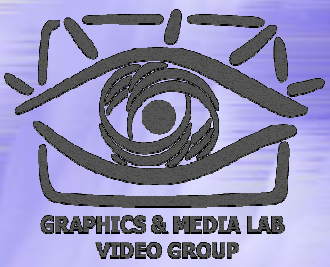
```
int tmp1, tmp2, i=0;
tmp1 = M[i];
while( tmp1 != x ) {
   tmp2 = tmp1;
   i++;
   tmp1 = M[i];
   M[i] = tmp2;
}
M[0] = x;
```

Обработаем **'рдакрааааабб'**:

Символ	Список	Выход
р	абдкр	4
д	рабдк	3
а	драбк	2
к	адрбк	4
р	кадрб	3
а	ркадб	2
а	аркдб	0
а	аркдб	0
а	аркдб	0
б	аркдб	4
б	баркд	0

Получим **'4324320040'**:

Метод MTF / Пример



Пусть есть цепочка: **'bbbbccccccddddddaaaaaab'**

Если сжимать ее по Хаффману, то вероятность всех символов ¼ и потребуется 20*2 = **40 бит**

Предположим, что начальный упорядоченный список символов выглядит как {'a', 'b', 'c', 'd'}.

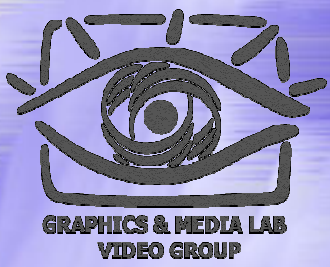
bbbbccccccddddddaaaaaab — исходная строка

10002000030000300003 — строка после MTF

Получившуюся строку можно упаковать по Хаффману в 15*1 + 3*2 + 3 + 3 = **27 бит**

Символ	Вероятность	Код Хаффмана
0	3/4	0
3	3/20	10
1	1/20	110
2	1/20	111

Метод MTF / Применение



- MTF наиболее эффективно применять на цепочках, получающихся после BWT.

  Общая схема алгоритма при этом выглядит как:
  BWT >> MTF >> RLE >> арифметич. сжатие

- Изредка MTF эффективен и просто перед словарными методами.

Структура материала

GRAPHICS & MEDIA LAB
VIDEO GROUP

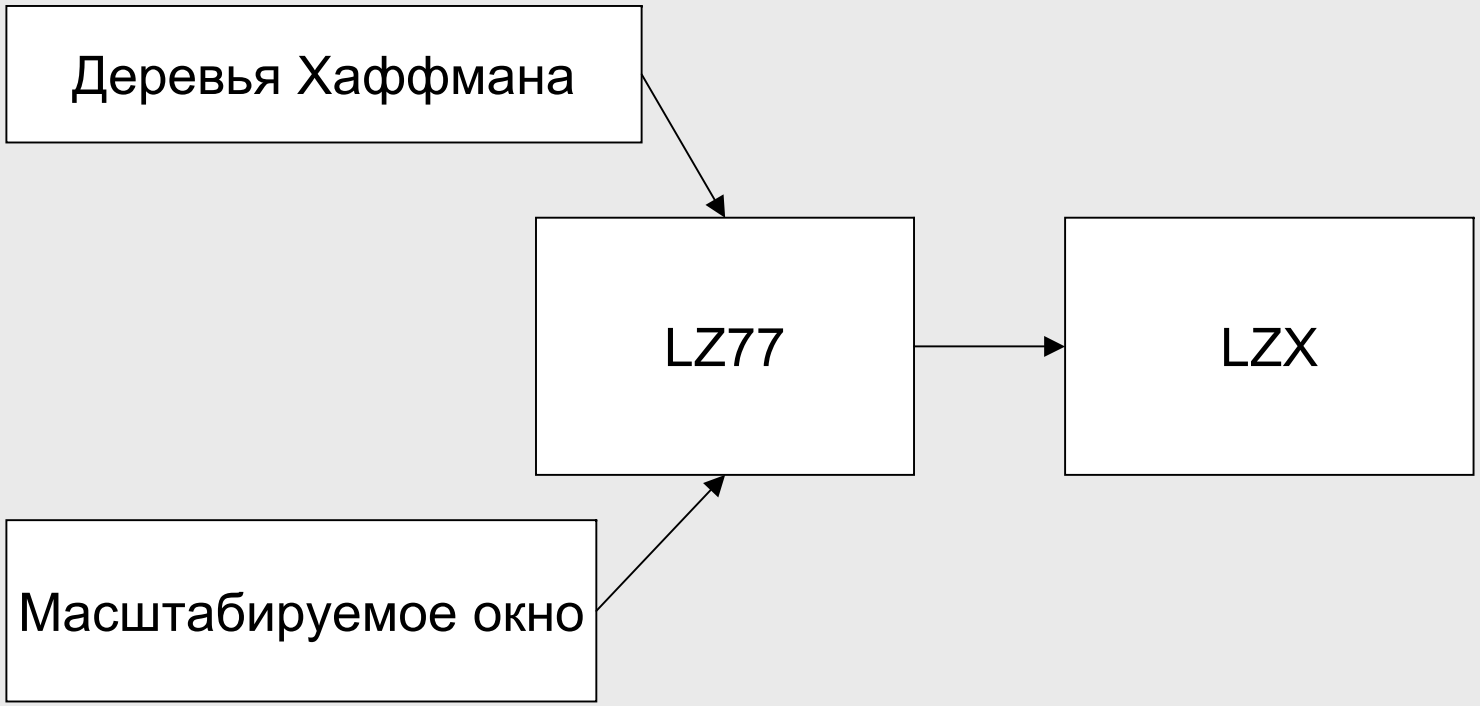
- Введение
  - Общие понятия сжатия
  - Теорема Шеннона
- Методы сжатия
  - Метод Хаффмана
  - Арифметическое сжатие
  - PPM
  - BWT (MTF)
  - **LZ-Huffman**

LZ-Huffman

GRAPHICS & MEDIA LAB
VIDEO GROUP

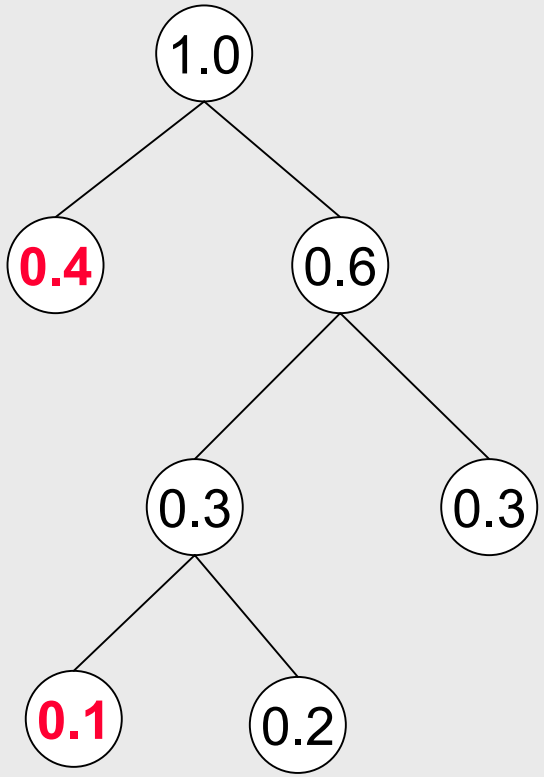
1. Основные идеи и понятия
   - Деревья Хаффмана
   - Repeated offsets
2. Алгоритм LZX
   - Предобработка
   - Сжатие информации
   - Типы блоков
   - Кодирование деревьев

Модификация LZ77



Деревья Хаффмана

1. Если два элемента имеют одинаковую длину пути, то элемент с меньшей частотой должен располагаться левее.



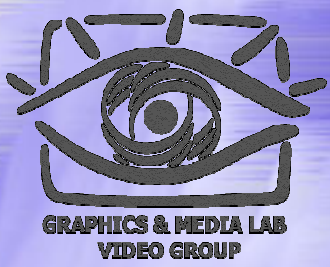
Деревья Хаффмана

GRAPHICS & MEDIA LAB
VIDEO GROUP

2.Если вершина имеет потомков, то все остальные вершины с той же длиной пути, лежащие левее, также должны иметь потомков.

3. Дерево должно содержать как минимум два элемента.

Деревья, используемые в алгоритме



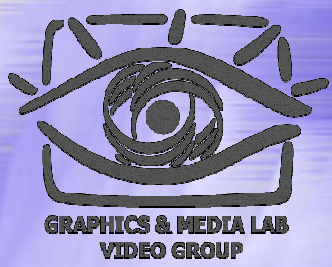
1. Основное дерево *(main tree)*.
2. Дерево длин *(length tree)*.
3. Дерево выровненных смещений *(aligned offset tree)*, pre-деревья *(pre-tree)*, etc.

LZ-Huffman

GRAPHICS & MEDIA LAB
VIDEO GROUP

1. Основные идеи и понятия
   - Деревья Хаффмана
   - Repeated offsets
2. Алгоритм LZX
   - Предобработка
   - Сжатие информации
   - Типы блоков
   - Кодирование деревьев

Repeated Offsets (LZ77 modifications)



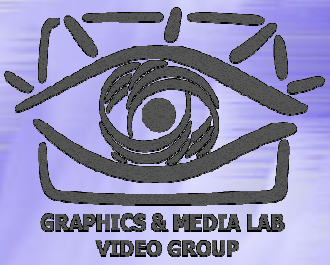
- Идея: отдельно хранить три наиболее часто употребляемых смещения (вернее их коды *(repeated offset codes)*) в отдельном списке.
- Структура списка :

R0 – самое последнее смещение

R1 – предпоследнее смещение

R2 – третье по счету.

Работа со списком смещений



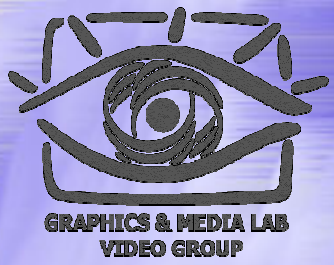
Match offset X where...	**Operation**
X ≠ R0 and X ≠ R1 and X ≠ R2	R2 ← R1 R1 ← R0 R0 ← X
X = R0	None
X = R1	swap R0 ⇔ R1
X = R2	swap R0 ⇔ R2

LZ-Huffman

GRAPHICS & MEDIA LAB
VIDEO GROUP

1. Основные идеи и понятия
   - Деревья Хаффмана
   - Repeated offsets
2. Алгоритм LZX
   - Предобработка
   - Сжатие информации
   - Типы блоков
   - Кодирование деревьев

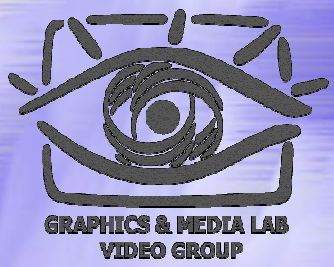
Алгоритм LZX



Формат данных:

Header	Block	Block	Block	…

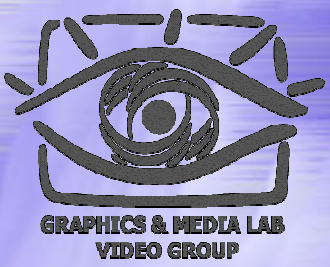
Алгоритм LZX



0		
1	Most significant 16 bits of file translation size	Least significant 16 bits of file translation size

Если первый бит равен 1, то имеется предварительная обработка. В таком случае за ним идет дополнительная информация о параметрах предварительного кодирования.

Содержание



1. Основные идеи и понятия
   - Деревья Хаффмана
   - Repeated offsets
2. Алгоритм LZX
   - Предобработка
   - Сжатие информации
   - Типы блоков данных
   - Кодирование деревьев

Предобработка

GRAPHICS & MEDIA LAB
VIDEO GROUP

Цель:

Предварительная обработка для улучшения сжатия 32х-разрядных исполняемых файлов (.exe, .dll, .ocx, …)

Предобработка

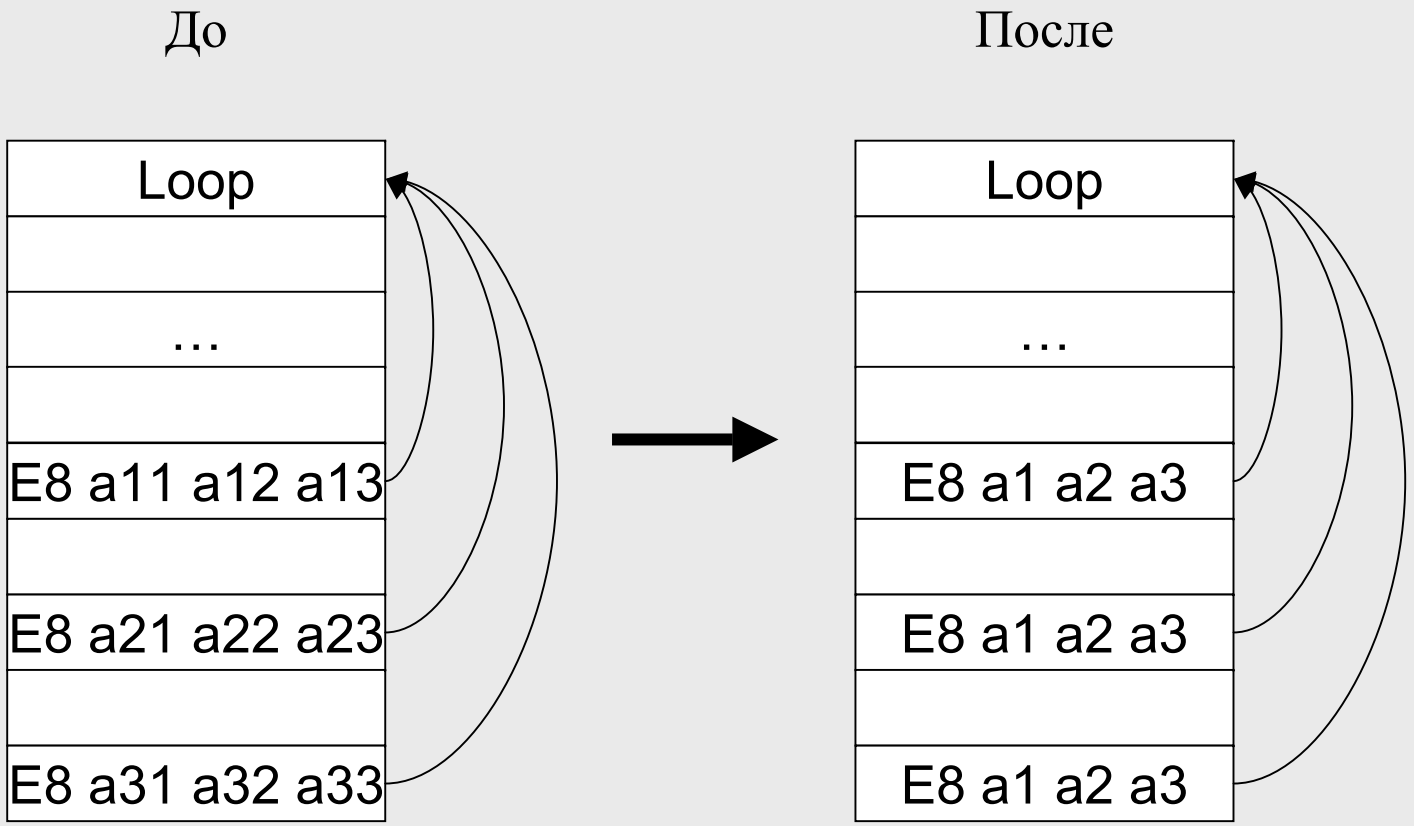
GRAPHICS & MEDIA LAB
VIDEO GROUP

Реализация:

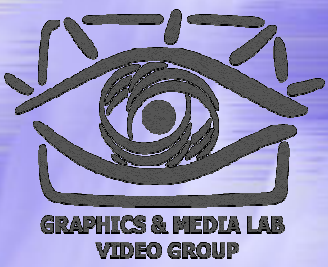
Замена во всех командах CALL (код E8h) относительного смещения на абсолютное.

Остальные данные не меняются.

Предобработка (Поясняющая диаграмма)



Предобработка (Pseudocode)



CALL byte sequence (E8 followed by 32 bit offset)

E8 r0 r1 r2 r3

Performing the relative-to-absolute conversion

```
relative_offset = r0 + (r1<<8) + (r2<<16) + (r3<<24);
new_value = conversion_function(current_location,
relative_offset);
a0 = bits_0_7(new_value);
a1 = bits_8_15(new_value);
a2 = bits_16_23(new_value);
a3 = bits_24_31(new_value);
```

Translated CALL byte sequence

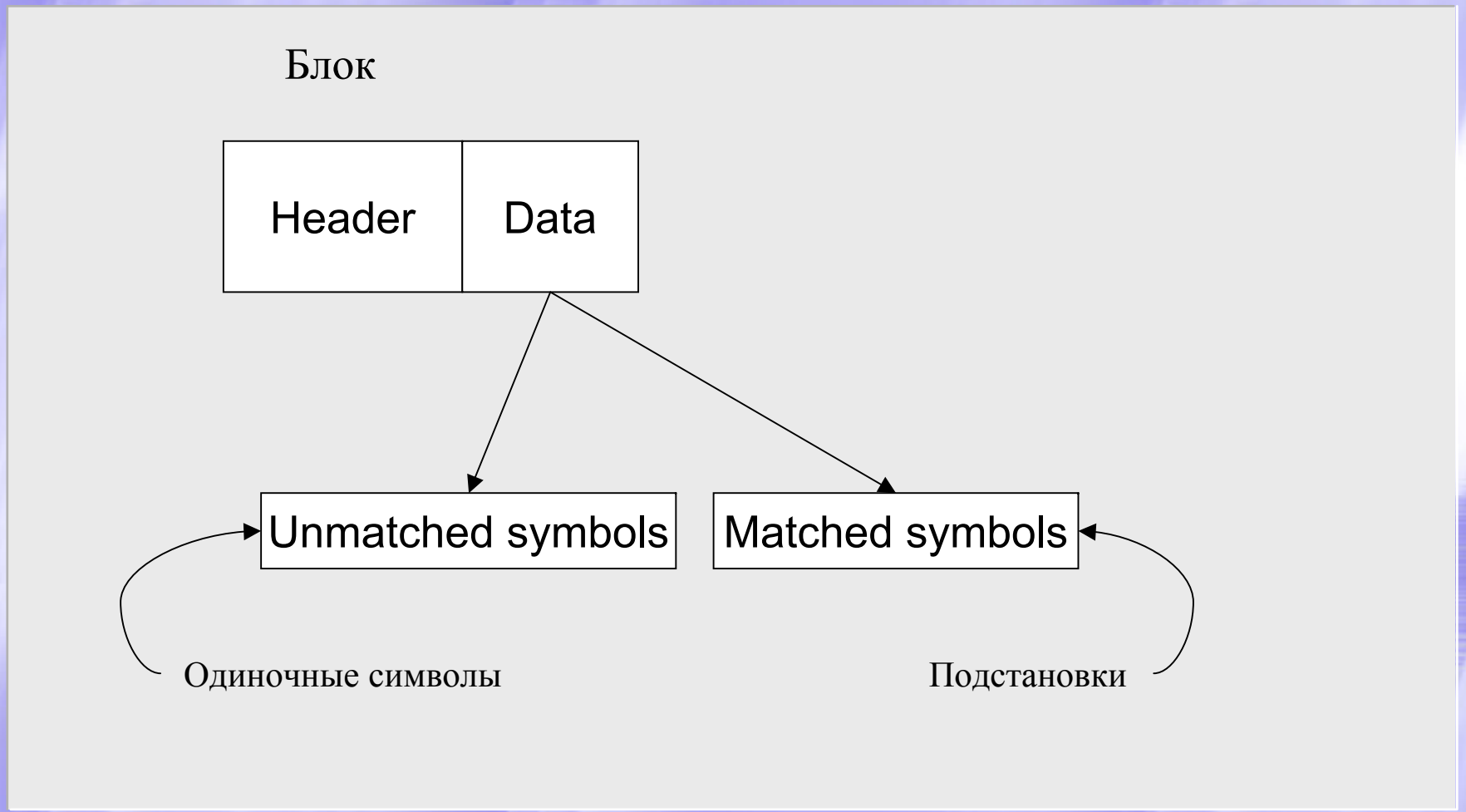
E8 a0 a1 a2 a3

LZ-Huffman

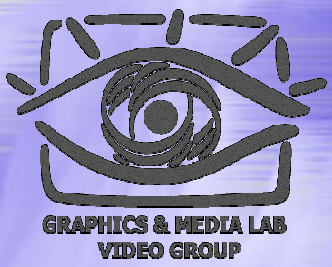
GRAPHICS & MEDIA LAB
VIDEO GROUP

1. Основные идеи и понятия
   - Деревья Хаффмана
   - Repeated offsets
2. Алгоритм LZX
   - Предобработка
   - Сжатие информации
   - Типы блоков данных
   - Кодирование деревьев

Сжатие информации (кодирование символов)

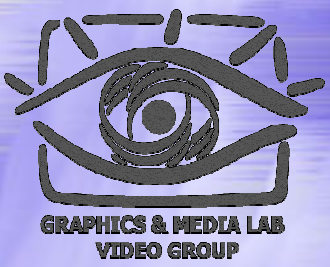


Одиночные символы (unmatched symbols)



Все 256 стандартных символов ASCII кодируются при помощи дерева Хаффмана. Что позволяет наиболее частые (для данного файла) символы кодировать меньшим количеством бит, а менее частые – б**о**льшим. Символы представляются элементами 0…(NUM_CHARS-1) основного дерева Хаффмана (main tree).

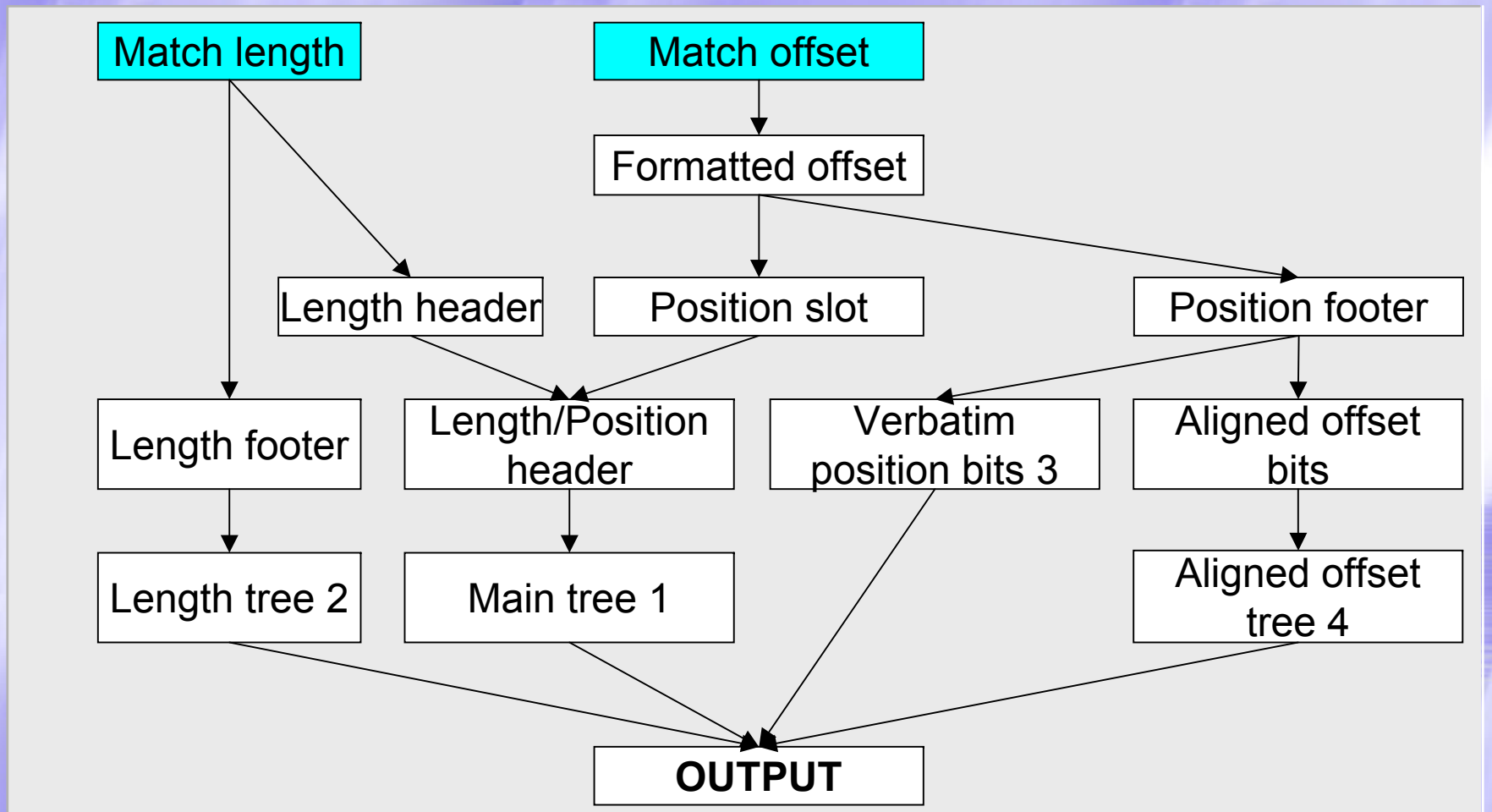
Кодирование подстановок



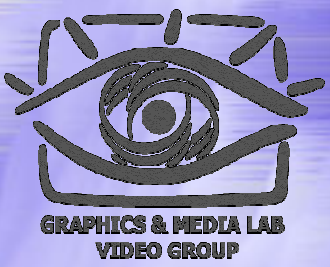
Идея:

Искать “большие” повторяющиеся последовательности. Записывать их один раз и давать им код. Далее ссылаться на них только по этому коду (несколько бит).

Кодирование подстановок



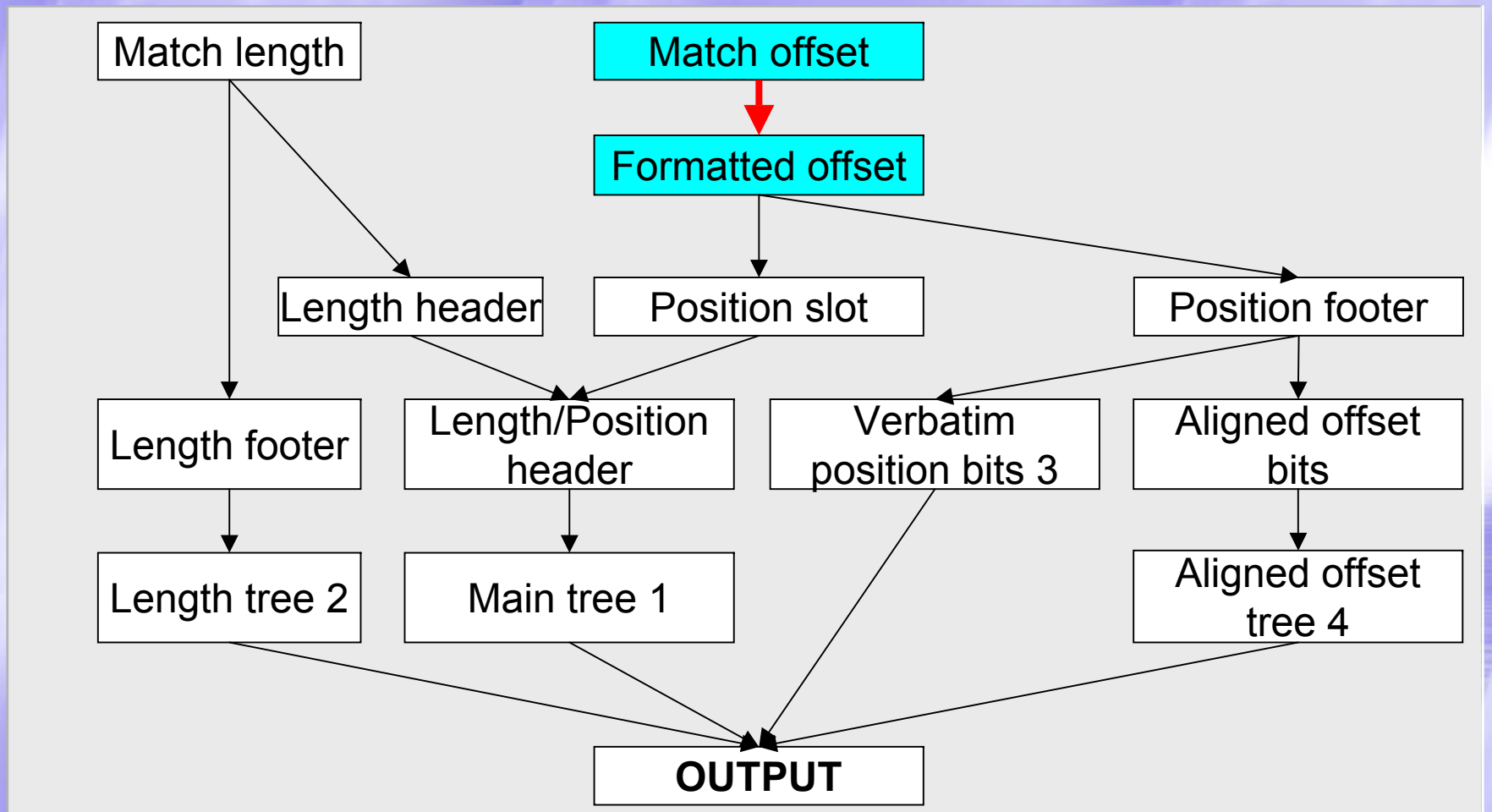
Кодирование подстановок



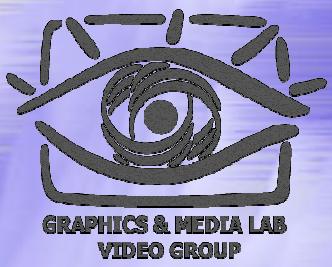
Подстановка задается двумя параметрами:

- длина подстановки (*match length*)
- смещение подстановки (*match offset*) относительно текущей позиции

Преобразование смещения



Преобразование смещения (Match offset ⇒ Formatted offset)



Converting a match offset to a formatted offset

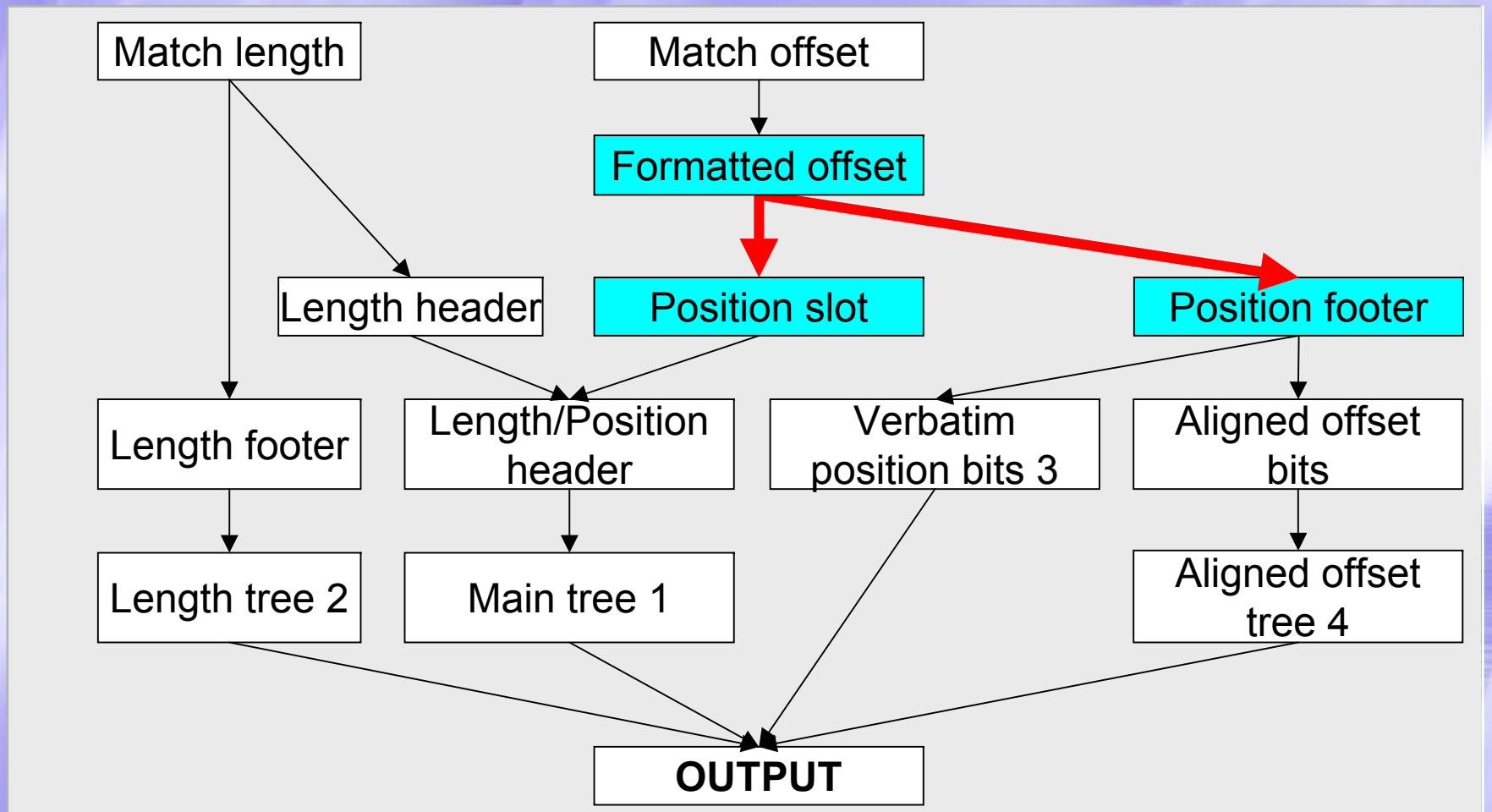
```
if (offset = = R0)
    formatted offset = 0;
else if (offset = = R1)
    formatted offset = 1;
else if (offset = = R2)
    formatted offset = 2;
else
    formatted offset = offset + 2;
```

Таблица смещений

GRAPHICS & MEDIA LAB
VIDEO GROUP

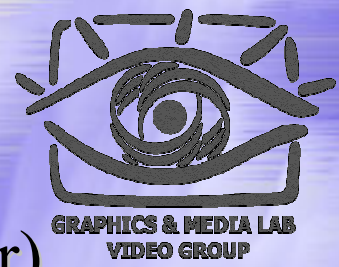
Encoded offset	**Real offset**
0	Most recent non-repeated match offset
1	Second most recent non-repeated match offset
2	Third most recent non-repeated match offset
3	1 (closest allowable)
4	2
...	...
500	498
x+2	X
WINDOW_SIZE-1 *(maximum possible)*	WINDOW_SIZE-3

Преобразование смещения



Преобразование смещения

(Formatted offset ⇒ Position slot, Position footer)



Форматированное смещение

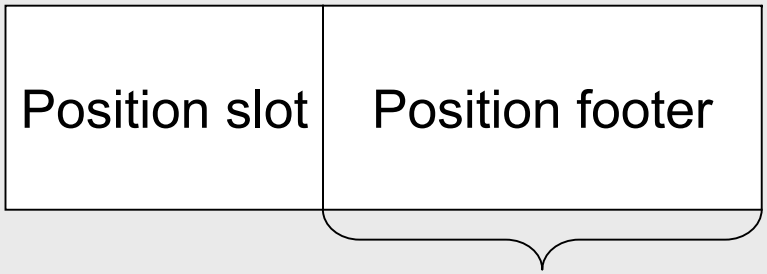
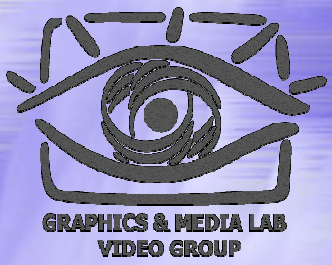
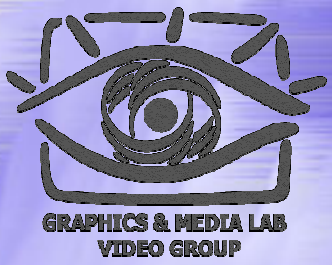


Таблица преобразования смещения



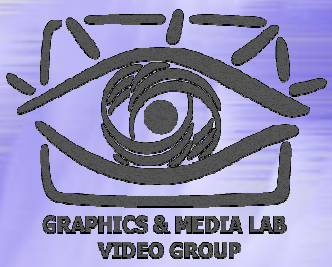
Position slot number	Base position	Number of position footer bits	Range of base position and position footer
0	0	0	0
1	1	0	1
2	2	0	2
3	3	0	3
4	4	1	4-5
5	6	1	6-7
6	8	2	8-11
7	12	2	12-15
8	16	3	16-23
...	...	...	...
48	1835008	17	1835008-1966079
49	1966080	17	1966080-2097151

Определение значения величины position footer



N (position slot)	extra_bits[n] (number of position footer bits)
0	0
1	0
2	0
3	0
4	1
5	1
6	2
7	2
8	3
...	...
35	16
36-49	17

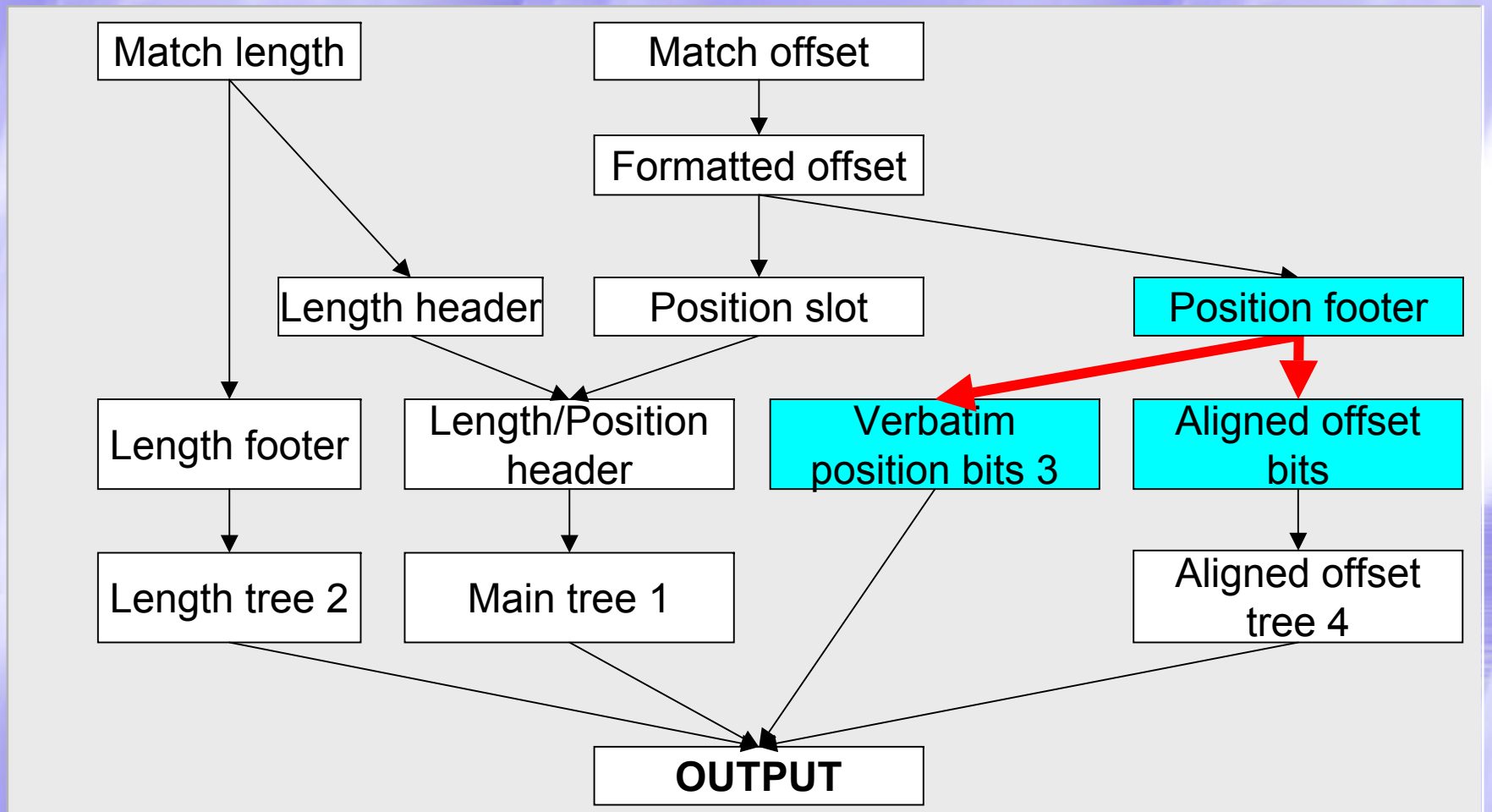
Вычисление значений position slot и position footer



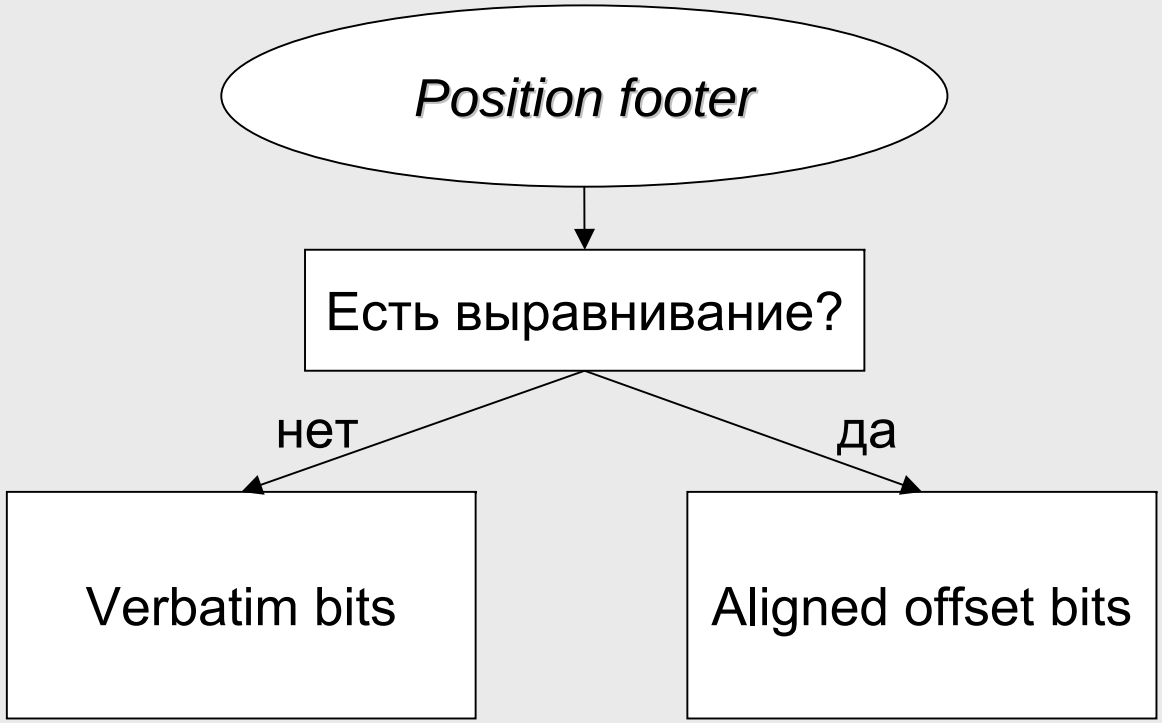
Calculating the position slot and position footer

```
position_slot =
   calculate_position_slot(formatted_offset);
position_footer_bits = extra_bits(position_slot);
if (position_footer_bits > 0)
     position_footer = formatted_offset &
                ((1<<position_footer_bits)-1);
else
   position_footer = null;
```

Position footer



Position footer

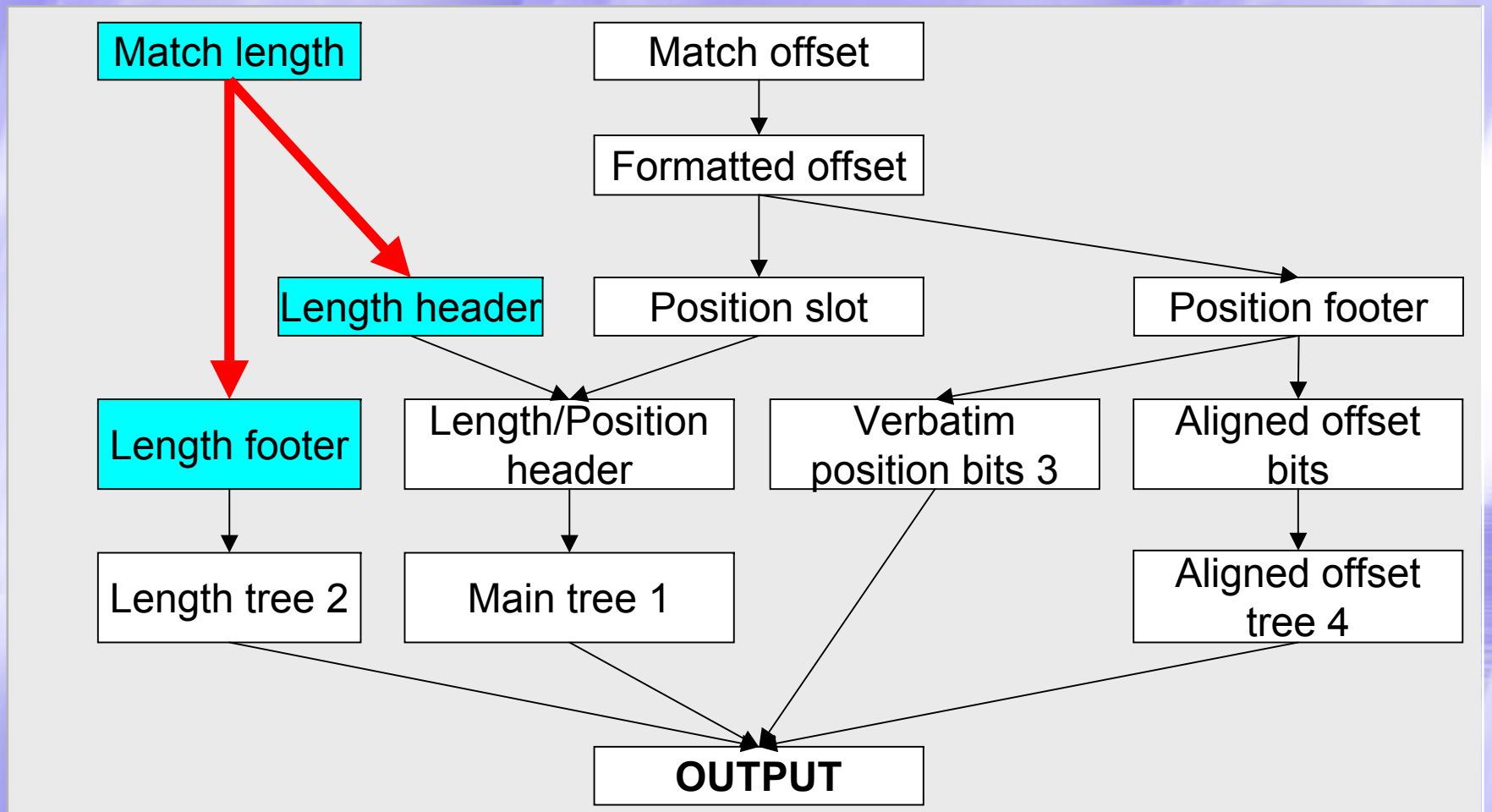


Position footer (code)

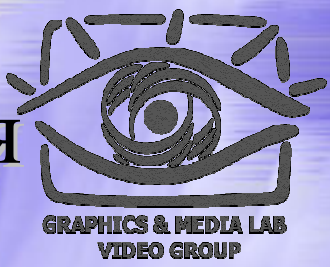
GRAPHICS & MEDIA LAB
VIDEO GROUP

```
if (block_type = = aligned_offset_block){
    if (formatted_offset <= 15){
        verbatim_bits = position_footer;
        aligned_offset = null;
    }
    else{
        aligned_offset = position_footer;
        verbatim_bits = position_footer >> 3;
    }
}
else{
    verbatim_bits = position_footer;
    aligned_offset = null;
}
```

Преобразование длины смещения



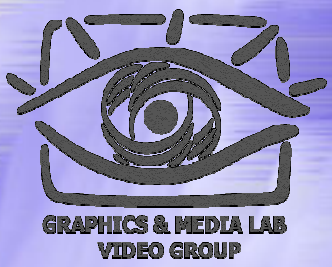
Преобразование длины смещения
(Match length ⇒ Length header, Length footer)



Pseudocode for obtaining the length header and footer

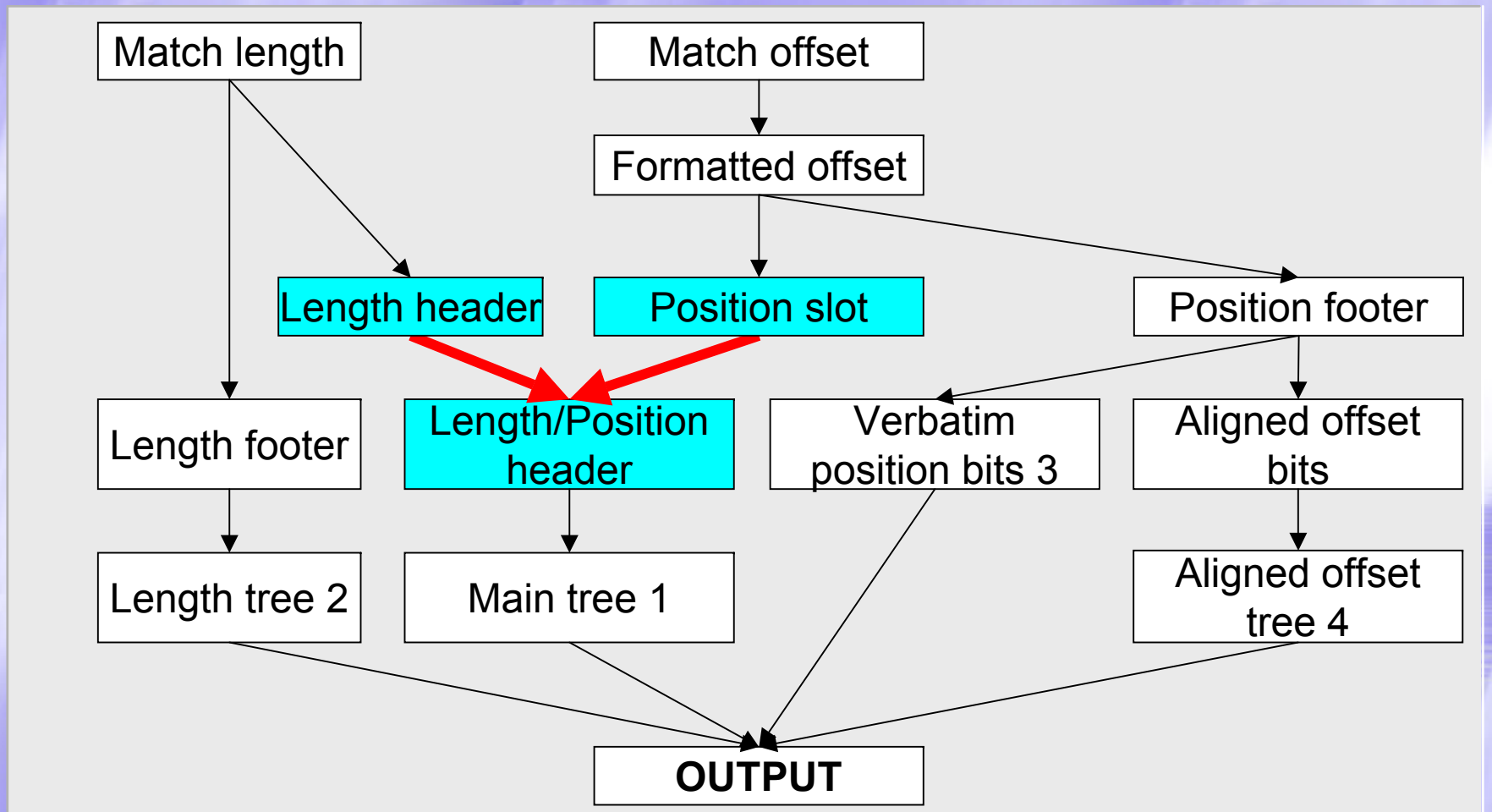
```
if (match_length <= 8){
    length_header = match_length-2;
    length_footer = null;
}
else{
    length_header = 7;
    length_footer = match_length-9;
}
```

Пример

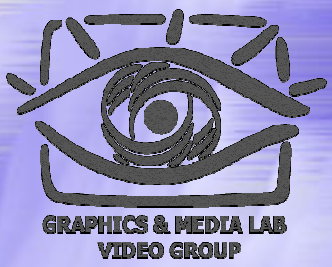


Match length	Length header	Length footer value
2 (MIN_MATCH)	0	None
3	1	None
4	2	None
5	3	None
6	4	None
7	5	None
8	6	None
9	7	0
10	7	1
50	7	41
257 (MAX_MATCH)	7	248

Diagram of match sub-components

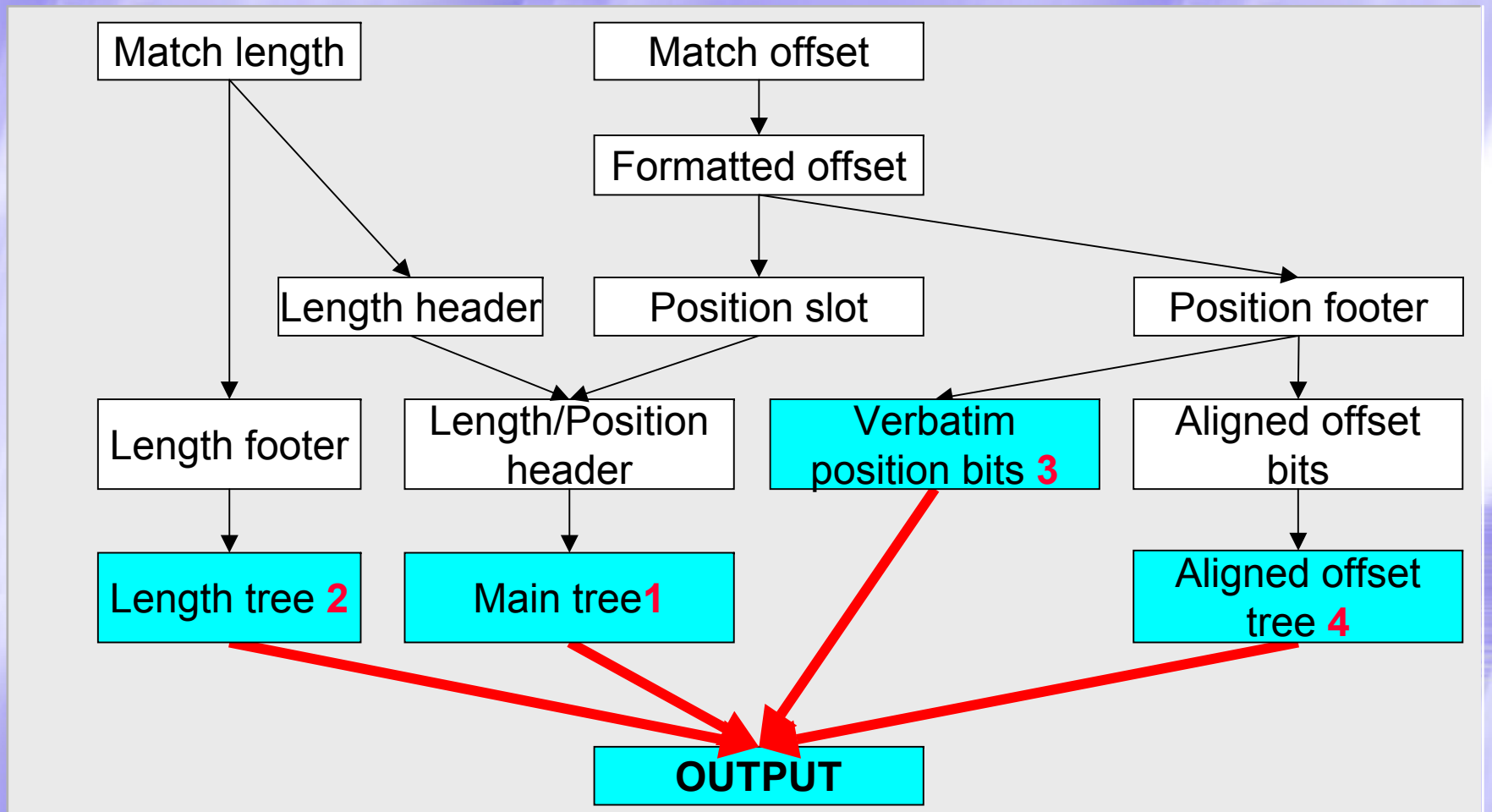


Length header, Position slot ⇒ Length / Position header

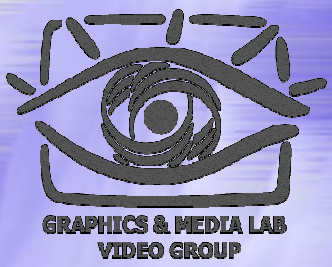


len_pos_header = (position_slot << 3) + length_header

Кодирование подстановки



Кодирование подстановки (Encoding a match)



Используется до 4-ех полей:

1. Output element (len_pos_header + NUM_CHARS) from the main tree
2. If length_footer != null, then output element length_footer from the length tree
3. If verbatim_bits != null, then output verbatim_bits
4. If aligned_offset_bits != null, then output element aligned_offset from the aligned offset tree

LZ-Huffman

GRAPHICS & MEDIA LAB
VIDEO GROUP

1. Основные идеи и понятия
   - Деревья Хаффмана
   - Repeated offsets
2. Алгоритм LZX
   - Предобработка
   - Сжатие информации
   - Типы блоков данных
   - Кодирование деревьев

Типы блоков

Первые три бита блока указывают, к какому типу он относится.

0	Undefined
1	Verbatim block
2	Aligned offset block
3	Uncompressed block
4-7	Undefined

не определено

сжатый блок

выровненные смещения

без сжатия

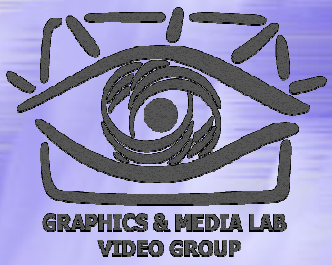
не определено

Блок без сжатия

1-16 bits	4 bytes	4 bytes	4 bytes	n bytes
zero padding	R0	R1	R2	Uncompressed data

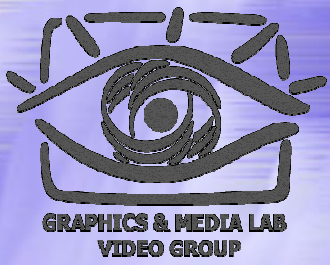
R0, R1, R2 – элементы списка смещений

Сжатый блок



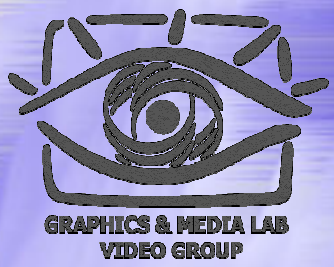
Entry	Comments	Size
Number of uncompressed bytes accounted for in this block	Range of $1...2^{24}$	24 bits
Pre-tree for first 256 elements of main tree	20 elements, 4 bits each	80 bits
Path lengths of first 256 elements of main tree	Encoded using pre-tree	Variable
Pre-tree for remainder of main tree	20 elements, 4 bits each	80 bits
Path lengths of remaining elements of main tree	Encoded using pre-tree	Variable
Pre-tree for length tree	20 elements, 4 bits each	80 bits
Path lengths of elements in length tree	Encoded using pre-tree	Variable
Compressed literals	Described later	Variable

Сжатый блок (диаграмма)



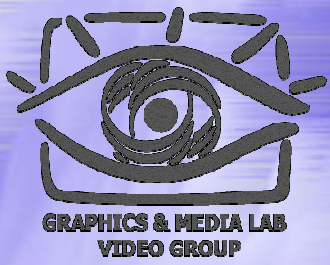
24 bits	80 bits	Variable	80 bits	Variable	80 bits	Variable	Variable
Size	Pre-tree for 256 elem of main tree	Path lengths for 256 elem of main tree	Pre-tree for remainder of main tree	Path lengths for remainder of main tree	Pre-tree for length tree	Path lengths for length tree	Compressed literals

Блок выровненных смещений



Entry	Comments	Size
Number of uncompressed bytes accounted for in this block	Range of $1...2^{24}$	24 bits
Pre-tree for first 256 elements of main tree	20 elements, 4 bits each	80 bits
Path lengths of first 256 elements of main tree	Encoded using pre-tree	Variable
Pre-tree for remainder of main tree	20 elements, 4 bits each	80 bits
Path lengths of remaining elements of main tree	Encoded using pre-tree	Variable
Pre-tree for length tree	20 elements, 4 bits each	80 bits
Path lengths of elements in length tree	Encoded using pre-tree	Variable
Aligned offset tree	8 elements, 3 bits each	24 bits
Compressed literals	Described later	Variable

Блок выровненных смещений (диаграмма)



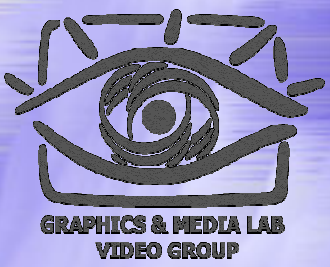
24 bits	**80 bits**	**Variable**	**80 bits**	**Variable**	**80 bits**	**Variable**	**24 bits**	**Variable**
Size	Pre-tree for 256 elem of main tree	Path lengths for 256 elem of main tree	Pre-tree for remainder of main tree	Path lengths for remainder of main tree	Pre-tree for length tree	Path lengths for length tree	Aligned offset tree	Compressed literals

LZ-Huffman

GRAPHICS & MEDIA LAB
VIDEO GROUP

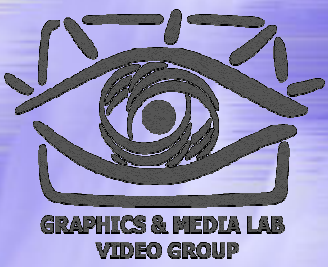
1. Основные идеи и понятия
   - Деревья Хаффмана
   - Repeated offsets
2. Алгоритм LZX
   - Предобработка
   - Сжатие информации
   - Типы блоков
   - Кодирование деревьев

Кодирование деревьев (Encoding of trees)



1. Основное дерево *(main tree)* кодируется в виде двух компонент:одно дерево для одиночных символов *(unmatched symbols)*, другое для подстановок *(matches)*.
2. С учетом ограничений на дерево Хаффмана, достаточно кодировать только длину пути *(path length)* каждого элемента.

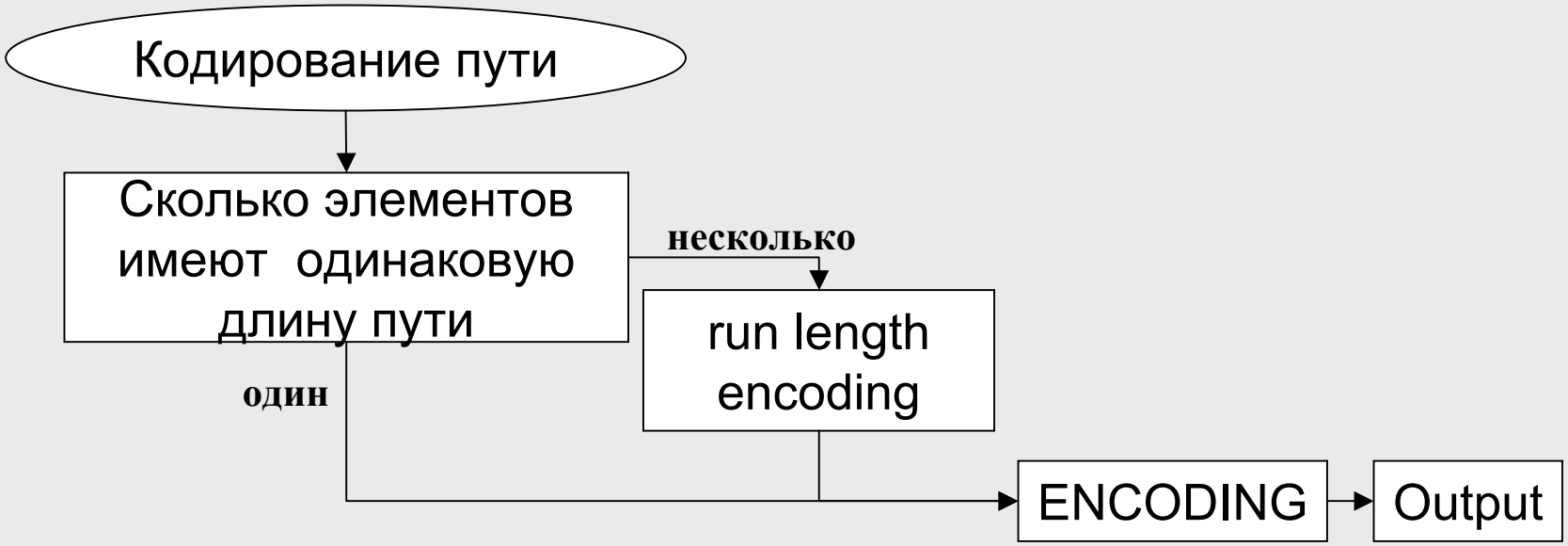
Кодирование деревьев (Encoding of trees)



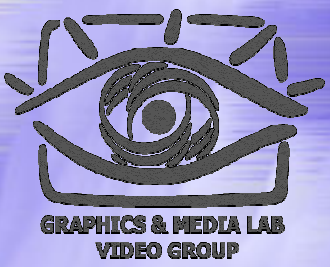
3. Для каждого блока – отдельное основное дерево:
   - невыгодно кодировать заново
   - лучше закодировать разницу между длиной пути в дереве первого блока и длиной пути в дереве следующего блока (для каждого элемента).

Кодирование деревьев (Encoding of trees)

Каждый элемент имеет длину пути от 0 до 16 включительно.



Кодирование длины пути (диаграмма)



Code	Operation
0-16	Len[x] = (prev_len[x] + code) mod 17
17	Zeroes = getbits(4) Len[x] = 0 for next (4 + Zeroes) elements
18	Zeroes = getbits(5) Len[x] = 0 for next (20 + Zeroes) elements
19	Same = getbits(1) Decode new Code Value = (prev_len[x] + Code) mod 17 Len[x] = Value for next (4 + Same) elements

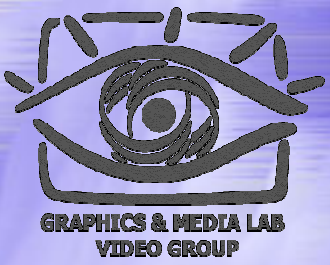
Кодирование длины пути (пояснение)

GRAPHICS & MEDIA LAB
VIDEO GROUP

- Коды 0-16 применяются, если только один элемент имеет соответствующую длину пути.
- Коды 17-19 применяются для дополнительного преобразования Run-Length Encoding.

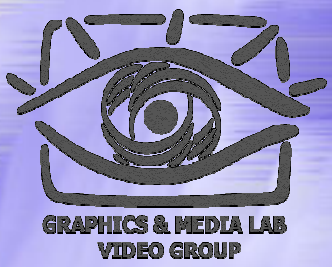
В результате получаем 20 кодовых элементов, которые можно закодировать 5 битами. Но здесь также применяется оптимизация (вспомогательные деревья или *pre-trees*)

Сжатие деревьев при помощи вспомогательных деревьев



20 кодов основного дерева кодируются в зависимости от частоты их появления при помощи вспомогательного дерева (pre-tree). Структура самого вспомогательного дерева не кодируется и имеет фиксированный размер 80 бит (20 элементов по 4 бита).

Вспомогательные деревья (pre-trees)



Length of tree code 0	4 bits
Length of tree code 1	4 bits
Length of tree code 2	4 bits
…	…
Length of tree code 18	4 bits
Length of tree code 19	4 bits

Задания

GRAPHICS & MEDIA LAB
VIDEO GROUP

- Задания по курсу расположены на странице курса: http://graphics.cs.msu.su/courses/mdc2004/