Д. Ватолин, А. Ратушняк,
М. Смирнов, В. Юкин

Методы сжатия данных

Приложения

Содержание книги:

ISBN 5-86404-170-X
2002

ПРИЛОЖЕНИЕ 1

```
/* Контекстный компрессор Dummy, автор М.Смирнов.
 * Исходный текст этой программы также можно найти
 * на сайте
 * http://compression.graphicon.ru/
 *
 * Применение:
 * e infile outfile  - закодировать infile в outfile
 * d infile outfile  - декодировать infile в outfile
 */

#include <stdio.h>

/* Класс для организации ввода/вывода данных */

class DFile {
   FILE *f;
public:
   int ReadSymbol (void) {
      return getc(f);
   };
   int WriteSymbol (int c) {
      return putc(c, f);
   };
   FILE* GetFile (void) {
      return f;
   }
   void SetFile (FILE *file) {
      f = file;
   }
} DataFile, CompressedFile;


/* Реализация range-кодера, автор Е.Шелвин
 * http://www.pilabs.org.ua/sh/aridemo6.zip
 */

typedef unsigned int  uint;

#define  DO(n)     for (int _=0; _<n; _++)
#define  TOP    (1<<24)

class RangeCoder
{
```

```
 uint code, range, FFNum, Cache;
 __int64 low; // Microsoft C/C++ 64-bit integer type
 FILE *f;

public:

 void StartEncode( FILE *out )
 {
   low=FFNum=Cache=0;  range=(uint)-1;
   f = out;
 }

 void StartDecode( FILE *in )
 {
   code=0;
   range=(uint)-1;
   f = in;
   DO(5) code=(code<<8) | getc(f);
 }

 void FinishEncode( void )
 {
   low+=1;
   DO(5) ShiftLow();
 }

 void FinishDecode( void ) {}

 void encode(uint cumFreq, uint freq, uint totFreq)
 {
   low += cumFreq * (range/= totFreq);
   range*= freq;
   while( range<TOP ) ShiftLow(), range<<=8;
 }

 inline void ShiftLow( void )
 {
   if ( (low>>24)!=0xFF ) {
      putc ( Cache + (low>>32), f );
      int c = 0xFF+(low>>32);
      while( FFNum ) putc(c, f), FFNum--;
      Cache = uint(low)>>24;
   } else FFNum++;
   low = uint(low)<<8;
 }
```

```
 uint get_freq (uint totFreq) {
   return code / (range/= totFreq);
 }

 void decode_update (uint cumFreq, uint freq, uint
totFreq)
 {
   code -= cumFreq*range;
   range *= freq;
   while( range<TOP ) code=(code<<8)|getc(f), range<<=8;
 }
} AC;

/* конец реализации range-кодера */


/* Структуры данных, глобальные переменные, константы */

struct ContextModel{
   int  esc,
      TotFr;
   int  count[256];
};

ContextModel cm[257],
             *stack[2];

int  context [1],
     SP;

const int MAX_TotFr = 0x3fff;


/* Собственно реализация компрессора */

void init_model (void){
   for ( int j = 0; j < 256; j++ )
      cm[256].count[j] = 1
   ;
   cm[256].TotFr = 256;
   cm[256].esc = 1;
   context [0] = SP = 0;
}

int encode_sym (ContextModel *CM, int c){
   stack [SP++] = CM;
```

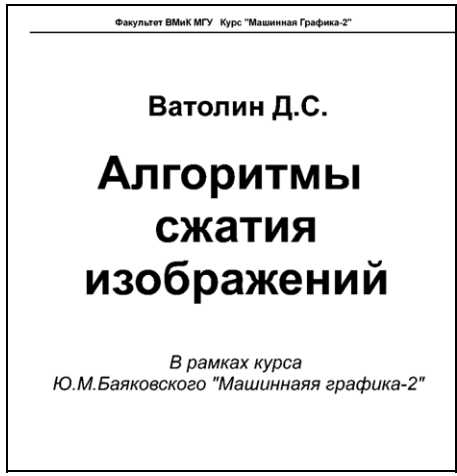
```
   if (CM->count[c]){
      int CumFreqUnder = 0;
      for (int i = 0; i < c; i++)
         CumFreqUnder += CM->count[i];
      AC.encode (CumFreqUnder, CM->count[c],
                 CM->TotFr + CM->esc);
      return 1;
   }else{
      if (CM->esc){
         AC.encode (CM->TotFr, CM->esc, CM->TotFr +
                    CM->esc);
      }
      return 0;
   }
}

int decode_sym (ContextModel *CM, int *c){
   stack [SP++] = CM;
   if (!CM->esc) return 0;

   int cum_freq = AC.get_freq (CM->TotFr + CM->esc);
   if (cum_freq < CM->TotFr){
      int CumFreqUnder = 0;
      int i = 0;
      for (;;){
         if ( (CumFreqUnder + CM->count[i]) <= cum_freq)
            CumFreqUnder += CM->count[i];
         else break;
         i++;
      }
      AC.decode_update (CumFreqUnder, CM->count[i],
                        CM->TotFr + CM->esc);
      *c = i;
      return 1;
   }else{
      AC.decode_update (CM->TotFr, CM->esc,
                        CM->TotFr + CM->esc);
      return 0;
   }
}

void rescale (ContextModel *CM){
   CM->TotFr = 0;
   for (int i = 0; i < 256; i++){
      CM->count[i] -= CM->count[i] >> 1;
      CM->TotFr += CM->count[i];
```

```
   }
}

void update_model (int c){
   while (SP) {
      SP--;
      if (stack[SP]->TotFr >= MAX_TotFr)
         rescale (stack[SP]);
      stack[SP]->TotFr += 1;
      if (!stack[SP]->count[c])
         stack[SP]->esc += 1;
      stack[SP]->count[c] += 1;
   }
}

void encode (void){
   int  c,
        success;
   init_model ();
   AC.StartEncode (CompressedFile.GetFile());
   while (( c = DataFile.ReadSymbol() ) != EOF) {
      success = encode_sym (&cm[context[0]], c);
      if (!success)
         encode_sym (&cm[256], c);
      update_model (c);
      context [0] = c;
   }
   AC.encode (cm[context[0]].TotFr, cm[context[0]].esc,
      cm[context[0]].TotFr + cm[context[0]].esc);
   AC.encode (cm[256].TotFr, cm[256].esc,
      cm[256].TotFr + cm[256].esc);
   AC.FinishEncode();
}

void decode (void){
   int  c,
        success;
   init_model ();
   AC.StartDecode (CompressedFile.GetFile());
   for (;;){
      success = decode_sym (&cm[context[0]], &c);
      if (!success){
         success = decode_sym (&cm[256], &c);
         if (!success) break;
      }
      update_model (c);
```

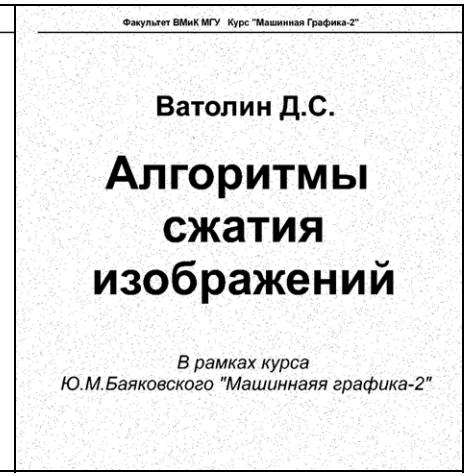
```
      context [0] = c;
      DataFile.WriteSymbol (c);
   }
}


void main (int argc, char* argv[]){
   FILE *inf, *outf;
   if (argv[1][0] == 'e'){
      inf = fopen (argv[2], "rb");
      outf = fopen (argv[3], "wb");
      DataFile.SetFile (inf);
      CompressedFile.SetFile (outf);
      encode ();
      fclose (inf);
      fclose (outf);
   }else if (argv[1][0] == 'd'){
      inf = fopen (argv[2], "rb");
      outf = fopen (argv[3], "wb");
      CompressedFile.SetFile (inf);
      DataFile.SetFile (outf);
      decode ();
      fclose (inf);
      fclose (outf);
   }
}
```

Приложение 2

Архивация двуцветного изображения



Изображение 1000x1000x2 цвета 125.000 байт



То же изображение с внесенными в него помехами

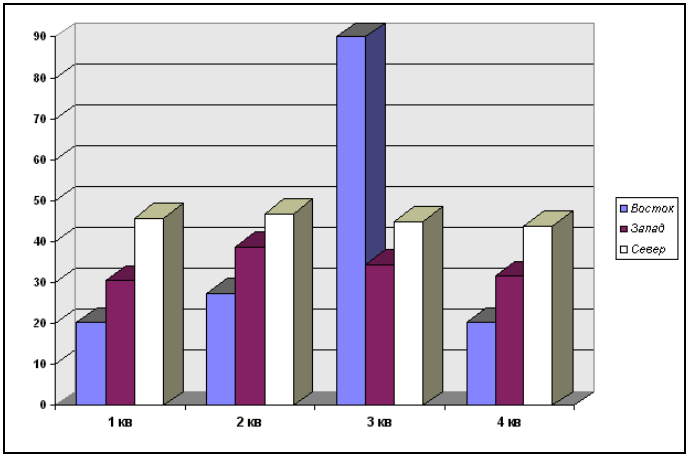
Ниже приведена степень компрессии изображений в зависимости от применяемого алгоритма:

	Алгоритм RLE	**Алгоритм LZW**	**CCITT Group 3**	**CCITT Group 4**
Без помех	10,6 (TIFF-CCITT RLE) 6,6 (TIFF-PackBits) 4,9 (PCX) 2,99 (BMP) 2,9 (TGA)	**12 (TIFF-LZW)** 10,1 (GIF)	9,5 (TIFF)	**31,2** **(TIFF)**
С помехами	5 (TIFF-CCITT RLE) 2,49 (TIFF-PackBits) 2,26 (PCX) 1,7 (TGA) 1,69 (BMP)	**5,4 (TIFF-LZW)** 5,1 (GIF)	4,7 (TIFF)	**5,12** **(TIFF)**

Выводы, которые можно сделать, анализируя данную таблицу:

1) Лучшие результаты показал алгоритм, оптимизированный для этого класса изображений CCITT Group 4 и модификация универсального алгоритма LZW.
2) Даже в рамках одного алгоритма велик разброс значений алгоритма компрессии. Заметим, что реализации RLE и LZW для TIFF показали заметно лучшие результаты, чем в других форматах. Более того, **во всех колонках все варианты** алгоритмов сжатия реализованные в формате TIFF лидируют.

Архивация 16-цветного изображения



Изображение 619x405x16 цвета 125.350 байт

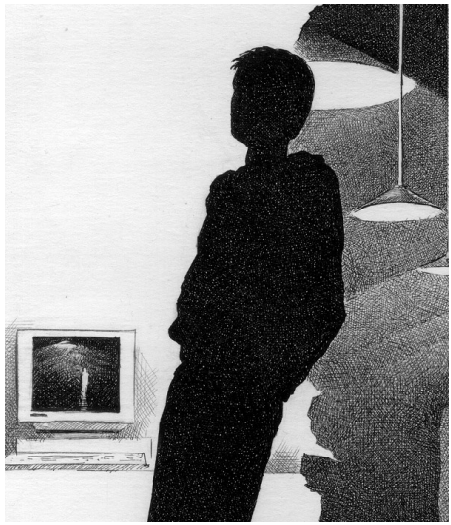
Ниже приведена степень компрессии изображений в зависимости от применяемого алгоритма:

	Алгоритм RLE	**Алгоритм LZW**
Первое изображение	5,55 (TIFF-PackBits) 5,27 (BMP) 4,8 (TGA) 2,37 (PCX)	**13,2 (GIF)** **11 (TIFF-LZW)**

Выводы, которые можно сделать, анализируя данную таблицу:

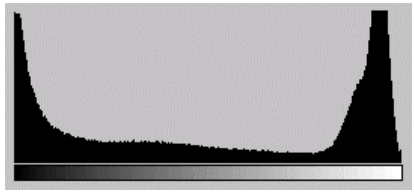
Не смотря на то, что данное изображение относится к классу изображений, на которые ориентирован алгоритм RLE (отвечает критериям «хорошего» изображения для алгоритма RLE), заметно лучшие результаты для него дает более универсальный алгоритм LZW.

Архивация изображения в градациях серого

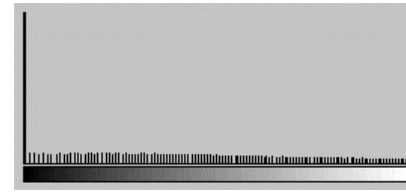


Изображение 600x700x256 градаций серого сразу после сканирования. 420.000 байт.

То же изображение с выровненной гистограммой плотности серого.



На гистограмме хорошо видны равномерные большие значения в области темных и «почти белых» тонов.



После выравнивания, пики есть только в значениях 0 и 255. В изображении присутствуют далеко не все значения яркости.

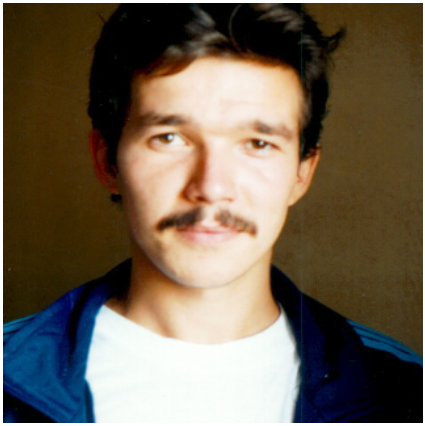
	Алгоритм RLE	**Алгоритм LZW**	**Алгоритм JPEG**
Оригинал	0,99 (TIFF-PackBits) 0,98 (TGA) 0,88 (BMP) 0,74 (PCX)	0,976 (TIFF-LZW) 0,972 (GIF)	**7,8 (JPEG q=10)** **3,7 (JPEG q=30)** 2,14 (JPEG q=100)
После обработки	2,86 (TIFF-PackBits) 2,8 (TGA) 0,89 (BMP) 0,765 (PCX)	3,02 (TIFF-LZW) 0,975 (GIF)*	**6,9 (JPEG q=10)** **3,7 (JPEG q=30)** 2,4 (JPEG q=100)

* Для формата GIF в этом случае можно получить изображение меньшего размера используя дополнительные параметры.

Выводы, которые можно сделать анализируя таблицу:

1) Лучшие результаты показал алгоритм сжатия с потерей информации. Для оригинального изображения только JPEG смог уменьшить файл. Заметим, что увеличение контрастности уменьшило степень компрессии при максимальном сжатии — врожденное свойство JPEG.
2) Реализации RLE и LZW для TIFF опять показали заметно лучшие результаты, чем в других форматах. Степень сжатия для них после обработки изображения возросла в 3 раза(!). В то время, как GIF, PCX и BMP и в этом случае увеличили размер файла.

Архивация полноцветного изображения



Изображение 320x320xRGB — 307.200 байт

Ниже приведена степень компрессии изображений в зависимости от применяемого алгоритма:

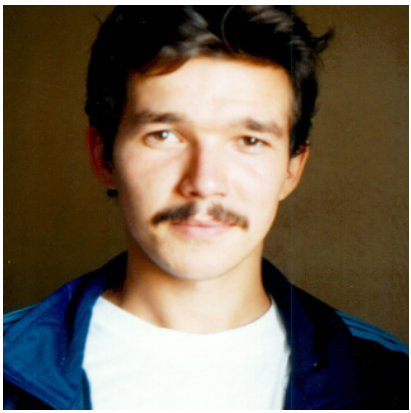
	Алгоритм RLE	**Алгоритм LZW**	**Алгоритм JPEG**
Первое изображение	1,046 (TGA) 1,037 (TIFF-PackBits)	1,12 (TIFF-LZW) 4,65 (GIF) *С потерями!* Изображение в 256 цветах	**47,2 (JPEG q=10)** **23,98 (JPEG q=30)** 11,5 (JPEG q=100)

Выводы, которые можно сделать, анализируя таблицу:

1) Алгоритм JPEG при визуально намного меньших потерях (q=100) сжал изображение в 2 раза сильнее, чем LZW с использованием перевода в изображение с палитрой.
2) Алгоритм LZW, примененный к 24-битному изображению практически на дает сжатия.
3) Минимальное сжатие, полученное алгоритмом RLE можно объяснить тем, что изображение в нижней части имеет

сравнительно большую область однородного белого цвета (полученную после обработки изображения).

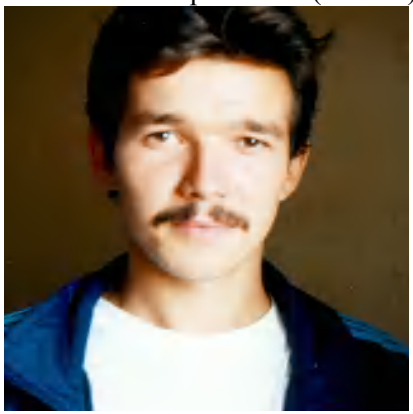
Архивация полноцветного изображения в 100 раз



320x320xRGB — 307.200 байт

Сжатие в 100 раз JPEG (3.08Kb)

Сжатие в 100 раз (3.04Kb) фрактальным алгоритмом



Сжатие в 100 раз (3.04Kb) wavelet алгоритмом

На данном примере хорошо видно, что при высоких степенях компрессии алгоритм JPEG оказывается полностью неконкурен-

тоспособным. Качество изображения для фрактального алгоритма визуально несколько ниже, однако для него не используется постобработка изображения (достаточно «разумное» сглаживание), из-за которого у волнового алгоритма размываются мелкие детали изображения.

Если вы обнаружите ошибки или неточности в данном тексте, просьба сообщить о них авторам по адресу:
compression@graphicon.ru

Ответы к вопросам и упражнениям и исходные тексты программ вы можете найти на
http://compression.graphicon.ru/