Ф.А. Мурзин, Д.Ф. Семич

ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ ТЕСТИРОВАНИЯ АЛГОРИТМОВ ПО ОБРАБОТКЕ ИЗОБРАЖЕНИЙ[1]

1. ВВЕДЕНИЕ

При создании прикладного программного обеспечения, ориентированного на обработку изображений, возникает необходимость в специальных программах, представляющих собой в некотором смысле “испытательные стенды”.

Такие программы должны иметь модульную структуру и развитый интерфейс, позволяющие легко подключать к ним новые программные блоки и отключать старые. Блоки, как правило, имеют конкретный содержательный смысл: фильтры, спектральные преобразования и т.д.

Реализованный в данном блоке алгоритм после испытаний на стенде и при установлении его полезности может быть включен в основные разрабатываемые программы в сильно измененном виде. Последнее обстоятельство связано с глубокой оптимизацией конечного программного продукта.

В данной работе кратко сообщается об алгоритмах, положенных в основу подобного “испытательного стенда”.

2. ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ ЭТАПЫ

2.1. Преобразование заданного в цветовом пространстве RGB изображения в систему координат YUV производится по формулам:

RGB −> YUV

$$Y = 0.114*B + 0.587*G + 0.299*R$$
$$U = 0.5000*B + 0.3313*G + 0.1687*R$$
$$V = 0.0813*B + 0.4187*G + 0.5000*R$$

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

YUV –> RGB

G = Y - 0.3441∗(U-128) - 0.7141∗(V-128)
B = Y + 1.772∗(U-128)
R = Y + 1.402∗(V-128)

Используются следующие приближения:

0.1145 = 3735/32768
0.2989 = 9798/32768
0.5866 = 19235/32768

0.5 = 16384/32768
0.3313 = 10855/32768
0.1687 = 5529/32768

0.0813 = 2264/32768
0.4187 = 13720/32768
0.5 = 16384/32768

1.772 = 14516/8192
1.402 = 11485/8192
0.3441 = 2819/8192
0.7141 = 5850/8192

2.2. Далее могут применяться различные дискретные интегральные преобразования: косинусное, Адамара, Хаара, Габора, Добеши, различные специфические вейвлет–преобразования типа 9/7 и другие. Например, дискретное косинусное преобразование (DCT) задается с помощью некоторой матрицы B и ее транспонированной матрицы B^T. Для блока размером 8×8 матрица B имеет вид:

$$B(i, j) = 0.5\,C \cos\{i\pi(2j+1)/16\},$$

где строки и столбцы имеют номера i и j соответственно, меняющиеся от 0 до 7, и

$$C = \begin{cases} 1/\sqrt{2} & \textit{if } i = 0, \\ 1 & \textit{otherwise}. \end{cases}$$

Заметим, что DCT является ортогональным преобразованием, и поэтому имеет место равенство $B^{-1} = B^T$.

Пусть f обозначает 8×8 блок изображения, и F — результат его преобразования. Блоки f и F могут быть получены один из другого посредством применения прямого (forward) (FDCT) и обратного (inverse) (IDCT) дискретных косинусных преобразований соответственно, которые задаются следующим образом

- $FDCT:\ F = BfB^T$,
- $IDCT:\ f = B^T FB$.

2.3. Квантование. На этом этапе вычисляется матрица квантования, в соответствии со следующим псевдокодом:

```
for(i=0;i<8;i++)
{
        for(j=0;j<8;j++)
        Q[i][j] = 1+((1+i+j)*q);
}
```

где q — коэффициент качества, от которого зависит степень потери качества сжатого изображения.
Для $q = 2$ имеем матрицу квантования:

$$Q = \begin{bmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{bmatrix}.$$

Квантование, описанное выше, называется грубым (coarse). В программе предусмотрено также тонкое (fine) квантование.

2.4. Один из интересных возникающих вопросов состоит в определении того, сколько коэффициентов, полученных в результате тех или иных преобразований, достаточно оставлять, чтобы не потерять качества изображения. Например, косинусное преобразование лучше применять к изображениям, заданным в координатах YUV. При этом для Y желательно сохранять все коэффициенты, лежащие в левом верхнем углу выше диагонали, и эле-

менты, лежащие на самой диагонали. Таким образом, получаем 36 коэффициентов. Остальные можно вычеркнуть, даже если они остаются после квантования, описанного выше. Для компонент U, V достаточно оставить по одному угловому коэффициенту. Всего получим 38 коэффициентов.

Преобразование Адамара лучше применять к изображениям, заданным в координатах RGB. При этом для каждой координаты достаточно оставлять 22 коэффициента, которые будут располагаться в левом верхнем углу в виде сектора круга. В итоге получим, что необходимо хранить $22 \times 3 = 66$ коэффициентов.

Квантование в этом случае применяется более простое, а именно: все числа делятся на одно и то же число, обычно на 4. Можно делить на 8 или 16, но при этом теряется качество.

С другой стороны, преобразование Адамара довольно хорошо себя ведет относительно разреживания исходного изображения и последующей интерполяции восстановленного изображения. Разреживание можно произвести по обеим пространственным координатам, т.е. в четыре раза. Учитывая тот факт, что коэффициентов необходимо хранить примерно в два раза больше, чем для косинусного преобразования, получаем, что разреживание дает выигрыш только в два раза, при этом происходит падение качества.

Таким образом, целесообразно не применять разреживание, а работать с косинусным преобразованием в координатах YUV.

2.5. Далее исследовался следующий метод компрессии компонент U, V. Рассмотрим дискретную функцию, характеризующую изменение спектрального коэффициента при движении вдоль блоков в горизонтальном направлении. Напомним, что для U, V оставляем только один коэффициент.

Данная функция определена в точках $0, \cdots, K = 720/8 - 1 = 89$ ввиду того, что исходное изображение имеет размер 720×480.

Аппроксимируем функцию ломаной линией по алгоритму, напоминающему алгоритм Брезенхема, известный в компьютерной графике. В отличие от классических интерполяционных задач, значения в конкретных точках аппроксимирующей функции могут отличаться от значений исходной дискретной функции, но на небольшую величину, которая задается и называется в программе Accuracy. С другой стороны, количество изломов должно быть как можно меньше (именно последнее обстоятельство важно для компрессии). Для уменьшения величин чисел будем запоминать расстояния между соседними точками, в которых наблюдаются изломы, и разности между соответствующими значениями аппроксимирующей ломаной.

2.6. На следующем этапе применяется LZW–метод кодирования. Такой алгоритм основан на динамически формируемом словаре, который состоит из различных строк. Строки заменяются кодами переменной длины.

В программе применяется один из алгоритмов, разработанных А.В. Кадачем*. Скорость компрессии этого метода — 20–25 Мбайт/с на Pentium–3 с тактовой частотой 500 Мгц, скорость декомпрессии — 80–120 Мбайт/с, степень компрессии — примерно такая же, как у PkZIP.

Описанная методика позволяет для U, V легко достигать степень компрессии в 350 раз.

2.7. В программе аналогично интерполируются старшие коэффициенты косинусного преобразования для компоненты Y. Предусмотрены три варианта: интерполируется один старший коэффициент, а также три или шесть коэффициентов.

2.8. Для высокочастотных коэффициентов компоненты Y такая методика не работает, так как данным коэффициентам соответствуют низкоамплитудные, но высокочастотные функции. Методика применима только к старшим коэффициентам.

2.9. По причине, указанной выше, в программе пока стоит “затычка”. В матрице коэффициентов размером 720×480 в каждом блоке размером 8×8 старшие коэффициенты зануляются: один, три или шесть соответственно (ранее они были уже проинтерполированы и закомпрессованы).

Полученная матрица младших коэффициентов компрессируется методом RLE. Были испробованы также другие методы: LZW, Huffman. Результаты были близкими и значительно зависящими от конкретного файла. Размер полученных файлов колебался от 1,6 до 30 Кбайт.

2.10. Более подробно опишем метод интерполяции (см. рисунок).

Для простоты будем считать, что задана непрерывная функция $f(x)$.

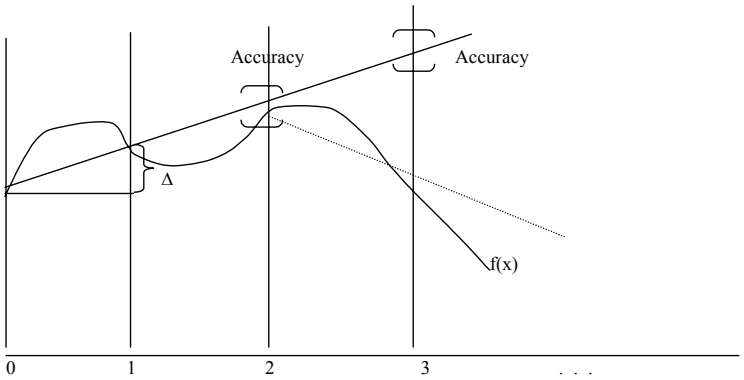
Вычисляем величину приращения $\Delta = f(1) - f(0)$. Соединим точки $f(0)$ и $f(1)$ отрезком прямой и продолжим данную прямую направо. Обозначим через $L(x)$ ее уравнение. Тогда легко видеть, что

$$L(2) = f(0) + 2\Delta = f(1) + \Delta,$$
$$L(3) = f(0) + 3\Delta = f(2) + \Delta.$$

Одновременно контролируем неравенство

* Кадач А. В. Эффективные алгоритмы неискажающего сжатия текстовой информации: Дис. канд. физ.-мат. наук: 05.13.11. — Новосибирск, 1997. — 186 с.

$$|f(k) - L(k)| \le \text{Accuracy}.$$



Величина Accuracy задает точность аппроксимации. Для компонент UV она может равняться 10 и более. Для Y ее желательно брать поменьше.

Таким образом будем продвигаться направо, пока не нарушится данное неравенство. В примере, показанном на рисунке, оно нарушается в точке $k = 3$. В этом случае отступим на шаг назад, запомним пару $(2, f(2))$, и алогичный процесс начнем сначала, т.е. будем искать новую Δ, и т.д. Продолжение процесса на рисунке изображено пунктирной линией.

Для хранения пар используются два различных массива: массив индексов и массив значений, которые дифференцируются, чтобы достичь дополнительной компрессии. Здесь, конечно, речь идет о разностной производной.

3. ЗАКЛЮЧЕНИЕ

Реализованная программа тестирования алгоритмов по обработке изображений показала свою полезность при разработке прикладного программного обеспечения. С помощью нее можно исследовать различные фильтры, алгоритмы скалирования, спектральные преобразования, варианты межкадрового сжатия для видео, методики вычисления векторов смещений и другие вопросы.

Программа реализована в системе Microsoft Visual C++ 6.0 с применением технологии MFC. Объем кода — 2.5 Мбайт.