Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики.

Кафедра вычислительной техники

Языки системного программирования

Лабораторная работа №7

Сепия фильтр для изображения

Выполнил:

Студент группы P3210

Глушков Дмитрий Сергеевич

Санкт-Петербург
2018 г.

Задание:
Используя SSE написать функцию на Ассемблере, реализующую сепия фильтр, замерить время выполнения функции, сравнить результаты с функцией, выполняющей аналогичное действие, но написанной на C.

Исходный код программы:

Sepia_c.c

```
#include <inttypes.h>
#include "bmp_lib.h"
#include "sepia_c.h"
static void sepia_one(pixel_t* const pixel);

void sepia_c_inplace(image_t* img )
{
  uint32_t x,y;
  for( y = 0; y < img->height; y++ )
    for( x = 0; x < img->width; x++ )
      sepia_one(&(img->data[y*img->width+x]));
}

static unsigned char sat( uint64_t x)
{
  if (x < 256)
    return x;
  return 255;
}

static void sepia_one(pixel_t* const pixel)
{
    static const float c[3][3] = {
            {.393f, .769f, .189f},
            {.349f, .686f, .168f},
            {.272f, .543f, .131f}};
    pixel_t const old = *pixel;
    pixel -> r = sat( old.r * c[0][0] + old.g * c[0][1] + old.b * c[0][2]
);
    pixel -> g = sat( old.r * c[1][0] + old.g * c[1][1] + old.b * c[1][2]
);
    pixel -> b = sat( old.r * c[2][0] + old.g * c[2][1] + old.b * c[2][2]
);
}
```

sepia_asm.asm

```
%define pix_arr r12
%define pixarr_size r13
%define res r14


times 4 db 0xff
align 16
c1_1: dd 0.272, 0.349, 0.393, 0.272
```

```
align 16
c2_1: dd 0.543, 0.686, 0.769, 0.543
align 16
c3_1: dd 0.131, 0.168, 0.189, 0.131

align 16
c1_2: dd 0.349, 0.393, 0.272, 0.349
align 16
c2_2: dd 0.686, 0.769, 0.543, 0.686
align 16
c3_2: dd 0.168, 0.189, 0.131, 0.168

align 16
c1_3: dd 0.393, 0.272, 0.349, 0.393
align 16
c2_3: dd 0.769, 0.543, 0.686, 0.769
align 16
c3_3: dd 0.189, 0.131, 0.168, 0.189

align 16
max_colvalues: dd 255, 255, 255, 255
section .text
global sepiaasm

sepiaasm:

    push r12
    push r13
    push r14
    mov pix_arr, rdi
    mov pixarr_size, rsi
    mov res, rdx
    movaps xmm6, [max_colvalues]

    test pixarr_size, pixarr_size
    jz .end
.fourpix:

        pxor xmm0, xmm0
        pxor xmm1, xmm1
        pxor xmm2, xmm2
        pinsrb xmm0, byte[pix_arr], 0
        shufps xmm0, xmm0, 0b00000000
        pinsrb xmm0, byte[pix_arr +3], 12
        cvtdq2ps xmm0, xmm0
        pinsrb xmm1, byte[pix_arr+ 1], 0
        shufps xmm1, xmm1, 0b00000000
        pinsrb xmm1, byte[pix_arr + 4], 12
        cvtdq2ps xmm1, xmm1

        pinsrb xmm2, byte[pix_arr + 2], 0
        shufps xmm2, xmm2, 0b00000000
        pinsrb xmm2, byte[pix_arr + 5], 12
        cvtdq2ps xmm2, xmm2
```

```
    movaps xmm3, [c1_1]
    movaps xmm4, [c2_1]
    movaps xmm5, [c3_1]

    mulps xmm0, xmm3
    mulps xmm1, xmm4
    mulps xmm2, xmm5

    addps xmm0, xmm1
    addps xmm0, xmm2

    cvtps2dq xmm0, xmm0

    pminsd xmm0, xmm6
        pextrb [res], xmm0, 0
        pextrb [res + 1], xmm0, 4
        pextrb [res + 2], xmm0, 8
        pextrb [res + 3], xmm0, 12

    add pix_arr, 3
    add res, 4
        pxor xmm0, xmm0
        pxor xmm1, xmm1
        pxor xmm2, xmm2
            pinsrb xmm0, [pix_arr], 0
            pinsrb xmm0, [pix_arr + 3], 8
        shufps xmm0, xmm0,  0b10100000
        cvtdq2ps xmm0, xmm0
           pinsrb xmm1, [pix_arr + 1], 0
           pinsrb xmm1, [pix_arr + 4], 8
       shufps xmm1, xmm1,  0b10100000
       cvtdq2ps xmm1, xmm1
           pinsrb xmm2, [pix_arr + 2], 0
           pinsrb xmm2, [pix_arr + 5], 8
       shufps xmm2, xmm2,  0b10100000
       cvtdq2ps xmm2, xmm2


    ;В соответсвии с марицей перехода получаем новые значения для пикселя
(как для 1 пикселя)
    movaps xmm3, [c1_2]
    movaps xmm4, [c2_2]
    movaps xmm5, [c3_2]
    mulps xmm0, xmm3
    mulps xmm1, xmm4
    mulps xmm2, xmm5
    addps xmm0, xmm1
    addps xmm0, xmm2
    cvtps2dq xmm0, xmm0

    pminsd xmm0, xmm6

        pextrb [res], xmm0, 0
        pextrb [res + 1], xmm0, 4
        pextrb [res + 2], xmm0, 8
```

```
        pextrb [res + 3], xmm0, 12
    add pix_arr, 3
    add res, 4
        pxor xmm0, xmm0
        pxor xmm1, xmm1
        pxor xmm2, xmm2
            pinsrb xmm0, byte[pix_arr], 0
            pinsrb xmm0, byte[pix_arr+3], 4
        shufps xmm0, xmm0,  0b00010101
        cvtdq2ps xmm0, xmm0
            pinsrb xmm1, byte[pix_arr + 1], 0
            pinsrb xmm1, byte[pix_arr + 4], 4
        shufps xmm1, xmm1, 0b00010101
        cvtdq2ps xmm1, xmm1
            pinsrb xmm2, byte[pix_arr + 2], 0
            pinsrb xmm2, byte[pix_arr + 5], 4
        shufps xmm2, xmm2,  0b00010101
        cvtdq2ps xmm2, xmm2

    movaps xmm3, [c1_3]
    movaps xmm4, [c2_3]
    movaps xmm5, [c3_3]

    mulps xmm0, xmm3
    mulps xmm1, xmm4
    mulps xmm2, xmm5

    addps xmm0, xmm1
    addps xmm0, xmm2
    cvtps2dq xmm0, xmm0
    pminsd xmm0, xmm6
        pextrb [res], xmm0, 0
        pextrb [res + 1], xmm0, 4
        pextrb [res + 2], xmm0, 8
        pextrb [res + 3], xmm0, 12

    add pix_arr, 6
    add res, 4
    sub pixarr_size, 4
    jnz .fourpix
.end:
    pop r14
    pop r13
    pop r12
    ret
```

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <sys/time.h>
#include <sys/resource.h>
```

```
#include <unistd.h>
#include <stdint.h>
#include "bmp_lib.h"
#include "sepia_c.h"
#include "sepia_asm.h"

int main()
{
  image_t *origin_p, *c_p, *asm_p;
  FILE *origin_f, *origin_fc, *sepia_c, *sepia_asm;
  char *origin_name, *c_name = "_output_c.bmp", *asm_name =
"_output_asm.bmp";
  uint32_t status;
  uint64_t size;
  struct rusage rusage_c, rusage_a;
      struct timeval start_c, start_a;
      struct timeval end_c, end_a;
      long res_c, res_a;

      printf("Enter file's name: ");
  origin_name = malloc (100 * sizeof(char));
  scanf("%s", origin_name);
      if ((origin_f = fopen(origin_name, "r+b")) == NULL)
      {
            puts ("File not found");
            return 1;
      }
      origin_fc = fopen(origin_name, "r+b");
      sepia_c = fopen(c_name, "w+b");
      sepia_asm = fopen(asm_name, "w+b");

      origin_p = (image_t*) malloc(sizeof(image_t));
      c_p = (image_t*) malloc(sizeof(image_t));
      asm_p = (image_t*) malloc(sizeof(image_t));

      if ((status = input_bmp(origin_f, c_p)) != 0)
print_error_in(status);
      if ((status = input_bmp(origin_fc, origin_p)) != 0)
print_error_in(status);

      size = origin_p -> width * origin_p -> height;
      asm_p -> height = origin_p -> height;
      asm_p -> width = origin_p -> width;
      asm_p -> data = malloc(size * sizeof(pixel_t));

      getrusage(RUSAGE_SELF, &rusage_c );
      start_c = rusage_c.ru_utime;
      sepia_c_inplace(c_p);
      getrusage(RUSAGE_SELF, &rusage_c );
      end_c = rusage_c.ru_utime;
      res_c = ((end_c.tv_sec - start_c.tv_sec) * 1000000L) +
end_c.tv_usec - start_c.tv_usec;
      puts("Sepia with C function");
      printf("Process took %ld microseconds\n", res_c);
      if (output_bmp(sepia_c, c_p))
```

```
            puts("Output error\n");
        printf("Resulting file: %s\n", c_name);


        getrusage(RUSAGE_SELF, &rusage_a);
        start_a = rusage_a.ru_utime;
        sepiaasm(origin_p->data, size, asm_p->data);
        getrusage(RUSAGE_SELF, &rusage_a);
        end_a = rusage_a.ru_utime;
    res_a = ((end_a.tv_sec - start_a.tv_sec) * 1000000L) + end_a.tv_usec -
start_a.tv_usec;


    puts("==================");
        puts("Sepia with ASM function");
        printf("Process took %ld microseconds\n", res_a);
        if (output_bmp(sepia_asm, c_p))
            puts("Output error\n");
        printf("Resulting file: %s\n", asm_name);

        puts("==================");
    printf("ASM func was faster by %ld microseconds\n", res_c - res_a);
        return 0;
}
```

Результат работы программы:

```
stud@spifmo:~/lab7$ ./main
Enter file's name: pics/2.bmp
Sepia with C function
Process took 408000 microseconds
Resulting file: _output_c.bmp
==================
Sepia with ASM function
Process took 44000 microseconds
Resulting file: _output_asm.bmp
==================
ASM func was faster by 364000 microseconds
```

Вывод:

В результате выполнения лабораторной работы написана функция на Ассемблере, использующая SSE, реализующая сепия фильтр, замерено время выполнения функции, проведено сравнение результатов с функцией, выполняющей аналогичное действие, но написанной на C. Правильно написанная ASM функция выполнялась быстрее.