Architektura komputerów 2
projekt

„Konwertowanie plików graficznych w formacie PPM do ciągu znaków ASCII odwzorowującego grafikę”

Janusz Długosz — 235746
Marcin Kotas — 235098

Prowadzący:
Mgr. Dominik Żelazny

Wrocław, 5 czerwca 2018

1 Założenia projektu

Plik PPM (ang. portable pixmap) – format zapisu grafiki rastrowej, w której nie używa się żadnej kompresji. Plik zawiera nagłówek z podstawowymi danymi – typ pliku, szerokość, wysokość i maksymalną wartość składową koloru. W dalszej części pliku znajdują się wartości składowe kolorów kolejnych pikseli oddzielone spacjami oraz znakami końca linii. Przykład:

```
P3
5 5
255
139 169 195 149 179 205 152 182 208 152 182 208 155 185 211
152 182 208 152 182 208 161 191 217 152 182 208 153 183 209
154 184 210 155 185 211 155 185 211 154 184 210 153 183 209
152 182 208 149 181 206 148 180 205 148 180 205 149 181 206
148 182 207 147 181 206 145 179 204 144 178 203 146 180 205
```

Program będzie wczytywał kolory, a później dzielił obraz na obszary o proporcjach odpowiadającym czcionką o stałej szerokości, a następnie obliczał średnią luminancję na danym segmencie i zamieniał ją na odpowiadający jej znak. Efekt końcowy powinien odwzorować obraz w stopniu pozwalającym na jego bezproblemową identyfikację. Po wykonaniu konwersji ciąg będzie zapisywany do pliku tekstowego.

2 Etap I

W pierwszym etapie stworzona została funkcja wczytująca dane z pliku oraz zapisująca je do trzech oddzielnych buforów (czerwony, zielony, niebieski). W osobnych zmiennych zapisywane są informacje definiujące szerokość oraz wysokość obrazu. Algorytm pomija linijki rozpoczynające się znakiem oznaczającego komentarz — '#'. Kolory kolejnych pikseli oddzielone są znakami białymi.

```
divide_by_color:
    push %rbp
    mov %rsp, %rbp
    mov 16(%rbp), %rsi  # %rsi - reg holding buf addr
    mov $0, %rdi        # %rdi - file_buf iterator
    mov $0, %r15        # %r15 - pixel colors iterator

    movw (%rsi, %rdi, 1), %bx   # PPM file begins with P3 number
    cmp $'P', %bl
    jne wrong_format
    cmp $'3', %bh
    jne wrong_format

    movq $3, %rdi   # skip 'P3\n'
    mov $0, %rcx
read_header:
    movb (%rsi, %rdi, 1), %bl
    inc %rdi
    cmp $'#', %bl       # skip comment lines
    je comment

    cmp $' ', %bl       # if reached space, width in to_numBuf
    je get_width
```

```
    cmp $'\n', %bl      # if reached new line, height in to_numBuf
    je get_height

    movb %bl, to_numBuf(, %rcx, 1) # read all width/height digits
    inc %rcx
    jmp read_header

get_width:
    push %rcx       # push number length
    push $to_numBuf # push buf addr
    call to_number
    pop width       # to_number returns by stack
    mov $0, %rcx
    jmp read_header # continue to read height

get_height:
    push %rcx
    push $to_numBuf
    call to_number
    pop height
    mov $0, %rcx
    add $4, %rdi    # skip color depth (assume 255)
    jmp read_pixel

comment:    # in loop until new line
    movb (%rsi, %rdi, 1), %bl
    inc %rdi
    cmp $'\n', %bl
    jne comment
    je read_header

read_pixel:
    read_red:
        movb (%rsi, %rdi, 1), %bl
        inc %rdi
        cmp $'\n', %bl  # skip new line (every 5 pixels)
        je read_red
        movb %bl, to_numBuf(, %rcx, 1)
        inc %rcx
        cmp $'0', %bl   # ' ', '\n' < '0'
        jge read_red    # (if greater, not finished reading number)

        dec %rcx        # don't include ' '
        push %rcx
        push $to_numBuf
        call to_number
        pop %rbx
        movb %bl, red(, %r15, 1)
        mov $0, %rcx
    read_green:
```

```
        movb (%rsi, %rdi, 1), %bl
        inc %rdi
        movb %bl, to_numBuf(, %rcx, 1)
        inc %rcx
        cmp $'0', %bl
        jge read_green

        dec %rcx
        push %rcx
        push $to_numBuf
        call to_number
        pop %rbx
        movb %bl, green(, %r15, 1)
        mov $0, %rcx
    read_blue:
        movb (%rsi, %rdi, 1), %bl
        inc %rdi
        movb %bl, to_numBuf(, %rcx, 1)
        inc %rcx
        cmp $'0', %bl
        jge read_blue

        dec %rcx
        push %rcx
        push $to_numBuf
        call to_number
        pop %rbx
        movb %bl, blue(, %r15, 1)
        mov $0, %rcx
    inc %r15
    cmp file_len, %rdi
    jl read_pixel

    mov %rbp, %rsp
    pop %rbp
ret

wrong_format:
    mov %rbp, %rsp
    pop %rbp
    add $8, %rsp    # skip ret addr to fix stack pointer
    jmp exit
```

3 Etap II

W typ etapie dodana została funkcja obliczająca średnią luminancję pikseli zawierających się w kolejnych prostokątach. Prostokąty mają proporcję 1:2 i posiadają zadaną szerokość.

Na początku obraz dzielony jest na prostokąty o wybranym rozmiarze. Część, która nadbywa jest ignorowana. Funkcja `getRect` przyjmuje jako argument ideks pierwszego piksela danego prostokąta. Następnie z tablic `red, green, blue` pobierane są składowe koloru danego piksela.

Wymnażane są one przez współczynniki według następującego wzoru:

$$L = \frac{2126 \cdot R + 7152 \cdot G + 722 \cdot B}{10000}$$

Na koniec funkcja liczy średnią luminancję dla całego prostokąta i zapisuje ją w tablicy `lum`.

Funkcja wywoływana jest w pętli w głównym programie. Pętla ta oblicza indeksy kolejnych prostokątów.

```
    mov width, %rax
    mov $0, %rdx
    divq fontWidth
    mov %rax, columnCount
    mov %rdx, ignore          # ignore division remainder
                              # (smaller than char font width)

    mov fontWidth, %rax
    mov $2, %rdi              # font height is 2 * font width
    mul %rdi                  # (1:2 proportions)
    mov %rax, fontHeight

    mov height, %rax
    mov $0, %rdx
    divq fontHeight
    mov %rax, rowCount

    mov fontHeight, %rax
    dec %rax
    mulq width
    add %rax, ignore     # skip pixels covered by font char

    mov $0, %r8      # current pixel index
    mov $0, %r9      # current row
    mov $0, %r11     # lum/char table iterator
nextRow:
    mov $0, %r10     # current column
    nextCol:
        call getRect
        add fontWidth, %r8
        inc %r10
        cmp columnCount, %r10
        jl nextCol
    add ignore, %r8
    inc %r9
    cmp rowCount, %r9
    jl nextRow
```

```
getRect:                  # r8 - buf index
    mov %r8, %rdi
    mov $10000, %r14      # factors were multiplied by 10000
    mov $0, %r15          # rect lum buffor
    mov $0, %rbx          # RectRows iterator
    RectRows:
        mov $0, %rcx      # cols iterator
```

```
        RectCols:
            mov $0, %rax
            movb red(, %rdi, 1), %al
            mov $lumR, %r12
            mul %r12
            mov %rax, %r13

            mov $0, %rax
            movb green(, %rdi, 1), %al
            mov $lumG, %r12
            mul %r12
            add %rax, %r13

            mov $0, %rax
            movb blue(, %rdi, 1), %al
            mov $lumB, %r12
            mul %r12
            add %r13, %rax
            mov $0, %rdx
            div %r14
            add %rax, %r15

            inc %rdi
            inc %rcx
            cmp fontWidth, %rcx
            jl RectCols
        sub fontWidth, %rdi      # return to first column
        add width, %rdi          # jump to next row
        inc %rbx
        cmp fontHeight, %rbx
        jl RectRows
    mov fontWidth, %rax          # calculate average luminance
    mulq fontHeight
    mov %rax, %rdi
    mov %r15, %rax
    mov $0, %rdx
    div %rdi

    mov %rax, lum(, %r11, 1)
    inc %r11
    ret
```

4 Etap III

Na podstawie wyliczonych luminancji dopasowywany jest znak ASCII według następującej skali:

```
"$@B%8&WM*oahkbdpqwmZOOQLCJUYXzvunxrjft/|()1{}]?-_+~<>i!lI:,^‘’. "
```

Luminancja w zakresie `0-255` dzielona jest przez 4, aby dopasować do skali 64 stopniowej. Następnie znak dobrany ze skali kopiowany jest do bufora wyjściowego `file_out`.

Bufor `file_out` zapisywany jest do pliku wyjściowego `out.txt`.

```
to_chars:
    mov $0, %rax
    mov $0, %r10
    movb lum(, %r8, 1), %al
    mov $0, %rdx
    div %r9
    mov scale(, %rax, 1), %r10b
    mov %r10b, file_out(, %rdi, 1)
    inc %rdi
    inc %r8
    inc %rsi
    cmp columnCount, %rsi
    jl to_chars_skip    # add new line if reached column count

    movb $'\n', file_out(, %rdi, 1)
    inc %rdi
    mov $0, %rsi

    to_chars_skip:
    cmp %r11, %r8
    jl to_chars

    mov %rdi, %r15      # file length

    movq $SYSOPEN, %rax
    movq $f_out, %rdi
    movq $O_WR_CRT_TRNC, %rsi
    movq $0666, %rdx
    syscall

    movq %rax, %rdi  # file handle
    movq $SYSWRITE, %rax
    movq $file_out, %rsi
    movq %r15, %rdx
    syscall

    movq $SYSCLOSE, %rax    # file handle still in %rdi
    syscall
```

5 Wyniki działania programu