САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №5

«OPEN MP»

Выполнил(а): Сентемов Лев Александрович

Номер ИСУ: 334863

студ. гр. М3135

Санкт-Петербург

2021

Цель работы: знакомство со стандартом OpenMP.

Инструментарий и требования к работе: C++, Стандарт OpenMP 2.0.

Алгоритм работы программы

1) Считываем PPM изображение из файла.
2) Для каждого возможного значения цвета каждого канала вычисляем количество пикселей изображения, принимающих это значение.
3) Находим минимальные и максимальные значения для каждого канала не учитывая долю самых светлых и самых темных точек.
4) Находим в каком из каналов самый большой диапазон
5) Растягиваем все значения цветов до необходимого диапазона

Листинг

main.cpp

```
#include <omp.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>
#include <cmath>

using namespace std;

struct ppm_file {
  unsigned int height;
  unsigned int width;
  unsigned char *image;
};
```

```
// Input
ppm_file read_ppm(char *input_file) {
  ifstream filein;
  filein.open(input_file);
  if (filein.is_open()) {
        std::string  P6;
        getline(filein, P6);
        unsigned int width, height, max_value;
        filein >> width >> height >> max_value;

        ppm_file input;
        input.height = height;
        input.width = width;
        input.image = new unsigned char[3 * height * width];
        filein >> input.image;
        filein.close();

        return input;
  } else {
        cout << "error";
  }
}


// Image processing
ppm_file process(ppm_file image, float coef) {
  unsigned int red_channel[256];
  memset(red_channel, 0, sizeof(red_channel));

  unsigned int green_channel[256];
  memset(green_channel, 0, sizeof(green_channel));

  unsigned int blue_channel[256];
```

```
	memset(blue_channel, 0, sizeof(blue_channel));

	unsigned int size = 3 * image.height * image.width;

	#pragma omp parallel for
	for (int i = 0; i < size; i++) {
		if (i % 3 == 0) {
			red_channel[(int) image.image[i]]++;
		} else if (i % 3 == 1) {
			green_channel[(int) image.image[i]]++;
		} else {
			blue_channel[(int) image.image[i]]++;
		}
	}

	int noise = round(size * coef);

	unsigned char r_min = 255, r_max = 0, g_min = 255, g_max = 0, b_min
= 255, b_max = 0;

	for (int i = 1; i < 256; i++) {
		red_channel[i] += red_channel[i - 1];
		green_channel[i] += green_channel[i - 1];
		blue_channel[i] += blue_channel[i - 1];

	for (int i = 0; i < 256; i++) {
		if (red_channel[i] >= noise) {
			r_min = min((unsigned char) i, r_min);
			break;
		}
	}

	for (int i = 0; i < 256; i++) {
		if (green_channel[i] >= noise) {
```

```
            g_min = min((unsigned char) i, g_min);
            break;
      }
}
for (int i = 0; i < 256; i++) {
      if (blue_channel[i] >= noise) {
            b_min = min((unsigned char) i, b_min);
            break;
      }
}

for (int i = 255; i > 0; i--) {
      if (red_channel[255] - red_channel[i - 1] >= noise) {
            r_max = max((unsigned char) i, r_max);
            break;
      }
}
for (int i = 255; i > 0; i--) {
      if (green_channel[255] - green_channel[i - 1] >= noise) {
            g_max = max((unsigned char) i, g_max);
            break;
      }
}
for (int i = 255; i > 0; i--) {
      if (blue_channel[255] - blue_channel[i - 1] >= noise) {
            b_max = max((unsigned char) i, b_max);
            break;
      }
}

unsigned char r_range = r_max - r_min;
unsigned char g_range = g_max - g_min;
unsigned char b_range = b_max - b_min;
```

```
  unsigned char old_min = r_min, old_max = r_max;
  if (r_range < g_range) {
        old_min = g_min;
        old_max = g_max;
        r_range = g_range;
  }

  if (r_range < b_range) {
        old_min = b_min;
        old_max = b_max;
        r_range = b_range;
  }
  if (r_range < 1)
        r_range = 1;
  float coef2 = 255.0 / ((float) r_range);

  #pragma omp parallel for
  for (int i = 0; i < size; i++) {
        if (image.image[i] <= old_min) {
              image.image[i] = 0;
        } else if (image.image[i] >= old_max) {
              image.image[i] = 255;
        } else if (i % 3 == 0) {
              image.image[i] = (unsigned char) round((image.image[i] -
r_min) * coef2);
        } else if (i % 3 == 1) {
              image.image[i] = (unsigned char) round((image.image[i] -
g_min) * coef2);
        } else {
              image.image[i] = (unsigned char) round((image.image[i] -
b_min) * coef2);
        }
  }
```

```
   return image;
}

// Output
void write_ppm(ppm_file image, char *output_file) {
   ofstream fout;
   fout.open(output_file, ios_base::out | ios_base::trunc);
   if (fout.is_open()) {
         fout << "P6\n" << image.width << ' ' << image.height <<
"\n255\n";
         fout << image.image;
         fout.close();
   } else {
         cout << "error";
   }
}



int main(int argc, char *argv[]) {
   stringstream convert{argv[1]};
   int th_num;

   if (!(convert >> th_num))
         th_num = 1;

   stringstream convert2{argv[4]};
   float coef;

   if (!(convert2 >> coef))
         coef = 0;

   omp_set_dynamic(0);
   omp_set_num_threads(th_num);
```

```
  // Input
  ppm_file image = read_ppm(argv[2]);

  // Image processing
  ppm_file result = process(image, coef);

  // Output
  write_ppm(result, argv[3]);

  return 0;
}
```