Tecnológico de Monterrey

Actividad 2.1

Iván Jesús Vázquez Rivera A01329911

Multiprocesadores

Introducción

Para esta actividad tenemos que leer imágenes BMP (Windows Bitmap) y usando malos, OpenMP y las diferentes librerías de C generar una imagen volteada horizontalmente y una volteada horizontal y verticalmente.

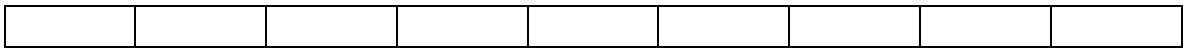
Implementación

Para resolver el problema pensamos en resolver diferentes puntos usando pequeñas tareas de forma secuencial; además, buscamos comprender más allá el funcionamiento de las imágenes en C.

Los archivos BMP tienen como característica que los primeros 54 registros especifican los datos de la imagen, diciendo el ancho en pixeles, el alto en pixeles, el tipo de archivos, entre otras cosas; después del encabezado viene toda la información de los valores de la imagen.

Para obtener esta información creamos un arreglo de 54 espacios de caracteres sin signo. Usando la función *fread* de C indicamos la fuente, el tamaño de cada registro a leer, el número de registros a leer y dónde se van a grabar. Con eso llenamos nuestro arreglo.

Usado la función *fseek* y *ftell* nos situamos en el registro final y preguntamos el número de registros en los archivos, de esta forma descubrimos que existe una relación entre los pixeles y el número de registros. Pensemos que cada imagen tiene un acho y un alto en pixeles. Usualmente en los registros la memoria se asigna de forma lineal, y cada uno de esos pixeles tendrá 3 registros por los valores de *blue*, *green* y *red*; por lo tanto, el número total de registros se vuelve la multiplicación del ancho por el alto por 3 más los 54 registros de encabezado.



Sabiendo esto generamos una estructura que contenta los valores de estos 3 registros. Para leer la imagen, creamos una matriz de esta estructura que va a leer la imagen usando *fgetc* que lee uno de los registros de un archivo aumento el puntero de índice en uno y almacenando el registro en una variable unsigned char

```
typedef struct{
    unsigned char b;
    unsigned char g;
    unsigned char r;
}pixel;
```

```c
unsigned char *header;
header = (unsigned char*)malloc(sizeof(unsigned char)*54);

fread(header, sizeof(unsigned char), 54, image);

int width = *(int*)&header[18];
int height = *(int*)&header[22];

printf("ancho %d alto %d\n", width, height);

pixel **pixels = NULL;
pixels = (pixel**)malloc(sizeof(pixel*)*height);
for (int i = 0; i < height; i++){
    pixels[i] = (pixel*)malloc(sizeof(pixel)*width);
    if(pixels[i] == NULL){
        printf("Falling malloc");
        exit(-1);
    }
}

fseek(image, 54, SEEK_SET);
for(int i=0; i<54; i++) fputc(header[i], final);
for(int i=0; i<54; i++) fputc(header[i], horizontal);
for(int i=0; i<54; i++) fputc(header[i], gs);
```

Es importe mencionar *fputc* que funciona al contrario de la lectura, para escribir un registro en un archivo.

Para convertir a escala de grises leemos los 3 registros de un pixel y usamos una relación matemática para obtener un solo valor entre 0 y 255 para después almacenar ese valor en las 3 variables de nuestra matriz.

```
void toGrayScale(pixel **pixels, int width, int height, FILE *image){
    unsigned char r, g, b;

    for (int i = 0; i < height; i++){
        for (int j = 0; j < width; j++){
            b = fgetc(image);
            g = fgetc(image);
            r = fgetc(image);

            unsigned char p = 0.21*r+0.72*g+0.07*b;

            pixels[i][j].b = p;
            pixels[i][j].g = p;
            pixels[i][j].r = p;
        }
    }
}
```

Para lograr la rotación en los dos ejes tenemos que crear un arreglo que almacene temporalmente los valores de una de las filas o columnas y los intercambie.

```
void changeHorizontal(pixel **m, int width, int height, int i){
    pixel * array;
    array = (pixel*)malloc(sizeof(pixel)*width);

    for (int j = 0; j < width; j++){
        array[j] = m[i][j];
    }

    for (int j = 0; j < width; j++){
        m[i][j] = array[width - j - 1];
    }

    free(array);
}

void changeVertical(pixel **m, int width, int height, int i){
    pixel * array;
    array = (pixel*)malloc(sizeof(pixel)*height);

    for (int j = 0; j < height; j++){
        array[j] = m[j][i];
    }

    for (int j = 0; j < height; j++){
        m[j][i] = array[height - j - 1];
    }

    free(array);
}
```

Estás operaciones pueden ser computadas totalmente paralelas ya que contiene un index y su acceso puede ser aleatorio sin causar perdida o comprometer la información

Resultados

Estos fueron los resultados obtenidos de una imagen con 700 pixeles y con una de 2000 pixeles

700

2000

```
#include <stdio.h>
#include <stdlib.h>
# include <math.h>
# include <omp.h>

#define NUM_THREADS 100

typedef struct{
   unsigned char b;
   unsigned char g;
   unsigned char r;
}pixel;

void printMatrix(pixel ** m, int width, int height);
void changeHorizontal(pixel **m, int width, int height, int i);
void changeVertical(pixel **m, int width, int height, int i);
void fillMatrix(pixel **m, int width, int height, FILE *file);
void toGrayScale(pixel **pixels, int width, int height, FILE *image);
void writeImage(pixel **pixels, int width, int height, FILE *image, int i);

int main(int argc, char const *argv[]){
   FILE *image, *gs, *horizontal, *final;
   image = fopen("sample1.bmp","rb");
   gs = fopen("gs.bmp", "wb");
   horizontal = fopen("horizontal.bmp", "wb");
   final = fopen("rp.bmp","wb");

   double t1, t2, tiempo;
```

```
omp_set_num_threads(NUM_THREADS);

unsigned char *header;
header = (unsigned char*)malloc(sizeof(unsigned char)*54);

fread(header, sizeof(unsigned char), 54, image);

int width = *(int*)&header[18];
int height = *(int*)&header[22];

printf("ancho %d alto %d\n", width, height);

pixel **pixels = NULL;
pixels = (pixel**)malloc(sizeof(pixel*)*height);
for (int i = 0; i < height; i++){
   pixels[i] = (pixel*)malloc(sizeof(pixel)*width);
   if(pixels[i] == NULL){
      printf("Falling malloc");
      exit(-1);
   }
}

fseek(image, 54, SEEK_SET);
for(int i=0; i<54; i++) fputc(header[i], final);
for(int i=0; i<54; i++) fputc(header[i], horizontal);
for(int i=0; i<54; i++) fputc(header[i], gs);

const double startTime = omp_get_wtime();
t1 = omp_get_wtime();
```

```
#pragma omp parallel
{
   #pragma omp single
      toGrayScale(pixels, width, height, image);

   #pragma omp barrier

   #pragma omp for ordered
      for(int i = 0; i < height; i++){
         #pragma omp ordered
            writeImage(pixels, width, height, gs, i);
      }

   #pragma omp for
      for (int i = 0; i < height; ++i){
            changeHorizontal(pixels, width, height, i);
      }

   #pragma omp barrier

   #pragma omp for ordered
      for(int i = 0; i < height; i++){
         #pragma omp ordered
            writeImage(pixels, width, height, horizontal, i);
      }

   #pragma omp for
      for (int i = 0; i < width; ++i){
```

```
            changeVertical(pixels, width, height, i);
        }

    #pragma omp for ordered
        for(int i = 0; i < height; i++){
            #pragma omp ordered
            writeImage(pixels, width, height, final, i);
        }
    }

    const double endTime = omp_get_wtime();

    t2 = omp_get_wtime();
    tiempo = t2 - t1;

    printf("tomo (%lf) segundos\n", tiempo);


    free(pixels);
    fclose(image);
    fclose(gs);
    fclose(horizontal);
    fclose(final);

    return 0;
}

void writeImage(pixel **pixels, int width, int height, FILE *file, int i){
    for (int j = 0; j < width; j++){
```

```
            fputc(pixels[i][j].b, file);
            fputc(pixels[i][j].g, file);
            fputc(pixels[i][j].r, file);
        }
    }

    void toGrayScale(pixel **pixels, int width, int height, FILE *image){
        unsigned char r, g, b;

        for (int i = 0; i < height; i++){
            for (int j = 0; j < width; j++){
                b = fgetc(image);
                g = fgetc(image);
                r = fgetc(image);

                unsigned char p = 0.21*r+0.72*g+0.07*b;

                pixels[i][j].b = p;
                pixels[i][j].g = p;
                pixels[i][j].r = p;
            }
        }
    }

    void printMatrix(pixel ** m, int width, int height){
        for (int i = 0; i < height; i++){
            for (int j = 0; j < width; j++){
                printf("b:%3u g:%3u r:%3u    ", m[i][j].b, m[i][j].g, m[i][j].r);
            }
```

```
        printf("\n");
    }
}

void fillMatrix(pixel **m, int width, int height, FILE *file){
    for (int i = 0; i < height; i++){
        for (int j = 0; j < width; j++){
            m[i][j].b = fgetc(file);
            m[i][j].g = fgetc(file);
            m[i][j].r = fgetc(file);
        }
    }
}

void changeHorizontal(pixel **m, int width, int height, int i){
    pixel * array;
    array = (pixel*)malloc(sizeof(pixel)*width);

    for (int j = 0; j < width; j++){
        array[j] = m[i][j];
    }

    for (int j = 0; j < width; j++){
        m[i][j] = array[width - j - 1];
    }

    free(array);
}
```

```
void changeVertical(pixel **m, int width, int height, int i){
    pixel * array;
    array = (pixel*)malloc(sizeof(pixel)*height);

    for (int j = 0; j < height; j++){
        array[j] = m[j][i];
    }

    for (int j = 0; j < height; j++){
        m[j][i] = array[height - j - 1];
    }

    free(array);
}
```