

BÀI TẬP VỀ NHÀ TUẦN 6

Xử lý ảnh và video

Đề: *Viết chương trình C thực hiện tính toán xem vật đã dịch chuyển bao nhiêu pixel về phía bên nào giữa 2 ảnh.*

1. PHẦN XÂY DỰNG MÃ NGUỒN C

- Khai báo các thư viện có sẵn trong C để nhập xuất giá trị, thư viện xử lý, đọc ghi cho ảnh BMP, thư viện các hàm toán học.

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

#include <math.h>
#include <string.h>
```

- Khởi tạo cấu trúc dữ liệu cho ảnh BMP, bao gồm thông tin của Header ảnh, thông tin chứa trong Header gồm có kích thước, số mảng màu, số bit màu cho 1 pixel, kiểu nén, số pixel theo mỗi cạnh...

```
#pragma pack(push, 1)
typedef struct {
    unsigned short type;
    unsigned int size;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int offset;
} BMPHeader;

typedef struct {
    unsigned int size;
    int width;
    int height;
    unsigned short planes;
    unsigned short bitsPerPixel;
    unsigned int compression;
    unsigned int imageSize;
```

```
int xPixelsPerMeter;
int yPixelsPerMeter;
unsigned int colorsUsed;
unsigned int importantColors;
} BMPInfoHeader;
#pragma pack(pop)
```

- Hàm khởi tạo bộ nhớ động cho ma trận 2 chiều và giải phóng bộ nhớ cho chúng sau khi xử lý xong.

```
unsigned char** MatrixMemory(int width, int height) {
    unsigned char **matrix = malloc(height * sizeof(unsigned char*));
    for (int i = 0; i < height; i++) {
        matrix[i] = malloc(width * sizeof(unsigned char));
    }
    return matrix;
}

void FreeMemory(unsigned char **matrix, int height) {
    for (int i = 0; i < height; i++) {
        free(matrix[i]);
    }
    free(matrix);
}
```

- Hàm đọc vào và lưu trữ dữ liệu cho ảnh màu RGB dạng BMP từ đường dẫn “filename”, đọc Header và lưu trữ Header cũng như giá trị của ảnh qua bộ nhớ động (biến *image).

```
unsigned char* ReadBMPimg(const char *filename, int *width, int *height) {
    FILE *fp = fopen(filename, "rb");
    if (fp == NULL) {
        printf("Không thể mở file %s\n", filename);
        exit(1);
    }
    BMPHeader bmpHeader;
    BMPInfoHeader bmpInfoHeader;
    fread(&bmpHeader, sizeof(BMPHeader), 1, fp);
    fread(&bmpInfoHeader, sizeof(BMPInfoHeader), 1, fp);
    if (bmpHeader.type != 0x4D42) {
        printf("Không phải ảnh BMP\n");
        fclose(fp);
        exit(1);
    }
    *width = bmpInfoHeader.width;
    *height = bmpInfoHeader.height;
    unsigned char *image = (unsigned char *)malloc((*width) * (*height) * 3);
```

```
fseek(fp, bmpHeader.offset, SEEK_SET);
fread(image, 3, (*width) * (*height), fp);
fclose(fp);
return image;
}
```

- Hàm ghi từ bộ nhớ động được xử lý ra ảnh dạng BMP vào đường dẫn filename, với các đặc tính tương ứng được thiết lập tương ứng.

```
void WriteGrayBMP(const char *filename, unsigned char **grayImage, int width,
int height) {
    FILE *fp = fopen(filename, "wb");
    if (fp == NULL) {
        printf("Không thể ghi ảnh tại %s\n", filename);
        exit(1);
    }

    BMPHeader bmpHeader;
    BMPInfoHeader bmpInfoHeader;
    bmpHeader.type = 0x4D42; // 'BM'
    bmpHeader.size = sizeof(BMPHeader) + sizeof(BMPInfoHeader) + width *
height;
    bmpHeader.reserved1 = 0;
    bmpHeader.reserved2 = 0;
    bmpHeader.offset = sizeof(BMPHeader) + sizeof(BMPInfoHeader);
    bmpInfoHeader.size = sizeof(BMPInfoHeader);
    bmpInfoHeader.width = width;
    bmpInfoHeader.height = height;
    bmpInfoHeader.planes = 1;
    bmpInfoHeader.bitsPerPixel = 8;
    bmpInfoHeader.compression = 0;
    bmpInfoHeader.imageSize = width * height;
    bmpInfoHeader.xPixelsPerMeter = 0;
    bmpInfoHeader.yPixelsPerMeter = 0;
    bmpInfoHeader.colorsUsed = 256;
    bmpInfoHeader.importantColors = 256;

    fwrite(&bmpHeader, sizeof(BMPHeader), 1, fp);
    fwrite(&bmpInfoHeader, sizeof(BMPInfoHeader), 1, fp);

    for (int i = 0; i < 256; i++) {
        unsigned char color[4] = {i, i, i, 0}; // B, G, R, A
        fwrite(color, sizeof(color), 1, fp);
    }

    for (int y = 0; y < height; y++) {
```

```
        fwrite(grayImage[y], 1, width, fp);
    }
    fclose(fp);
}
```

- Hàm chuyển đổi ảnh từ dạng màu RGB sang GrayScale, giá trị ảnh GrayScale lưu trữ dạng bộ nhớ động 2 chiều.

```
void RGB2Gray(unsigned char *rgbImage, unsigned char **grayImage, int width,
int height) {
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            unsigned char r = rgbImage[(y * width + x) * 3 + 2]; // R
            unsigned char g = rgbImage[(y * width + x) * 3 + 1]; // G
            unsigned char b = rgbImage[(y * width + x) * 3];      // B
            grayImage[y][x] = (unsigned char)(0.299 * r + 0.587 * g + 0.114 *
b);
        }
    }
}
```

- Hàm xác định cạnh Edge Detection sử dụng kernel lọc Sobel, đọc vào dạng ảnh GrayScale với chiều rộng, chiều cao tương ứng, trả về gradient là tổng hợp của xác định cạnh theo phương ngang và dọc theo dạng ma trận 2 chiều.

```
void SobelKernel(unsigned char **image, int width, int height, unsigned char
**gradient) {
    int x, y, i, j;
    int gx, gy;
    int kernel_x[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
    int kernel_y[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};

    for (y = 1; y < height - 1; y++) {
        for (x = 1; x < width - 1; x++) {
            gx = gy = 0;
            for (i = -1; i <= 1; i++) {
                for (j = -1; j <= 1; j++) {
                    unsigned char gray = image[y + i][x + j];
                    gx += gray * kernel_x[i + 1][j + 1];
                    gy += gray * kernel_y[i + 1][j + 1];
                }
            }
            gradient[y][x] = (unsigned char)fmin(sqrt(gx * gx + gy * gy),
255);
        }
    }
}
```

```
}  
}
```

- Hàm Histogram với chức năng phân tách vật thể với nền, ảnh đưa vào là gradient tạo từ hàm trên, phân tách qua giá trị th là Threshold, nếu giá trị pixel tại vị trí bất kỳ lớn hơn giá trị threshold (thường là của vật thể) thì đặt giá trị pixel tại đó là 255 (giá trị tối đa), nhỏ hơn thì là 0.

```
void Histogram(unsigned char **grayImage, unsigned char **Object, int width,  
int height, int th) {  
    for (int y = 0; y < height; y++) {  
        for (int x = 0; x < width; x++) {  
            if (grayImage[y][x] > th) {  
                Object[y][x] = 255;  
            } else {  
                Object[y][x] = 0;  
            }  
        }  
    }  
}
```

- Hàm xác định vị trí tâm của vật thể để xác định khoảng cách di chuyển, đưa vào ảnh Image với độ rộng và cao, xuất ra chỉ số vị trí của tâm, chỉ xét với những pixel có giá trị lớn hơn threshold.

```
void FindCenter(unsigned char **Image, int width, int height, int *centerX,  
int *centerY, int th) {  
    int totalX = 0, totalY = 0, count = 0;  
    for (int y = 0; y < height; y++) {  
        for (int x = 0; x < width; x++) {  
            if (Image[y][x] > th) {  
                totalX += x;  
                totalY += y;  
                count++;  
            }  
        }  
    }  
    if (count > 0) {  
        *centerX = totalX / count;  
        *centerY = totalY / count;  
    } else {  
        *centerX = -1;  
        *centerY = -1;  
    }  
}
```

- Hàm xác định sự di chuyển của vật thể giữa 2 ảnh Image1 và Image2 qua xét sự dịch chuyển của tâm 2 ảnh qua hàm tìm vị trí trung tâm phía trên và xét hướng và khoảng cách di chuyển, đồng thời xuất kết quả.

```
void MotionDetect(unsigned char **Image1, unsigned char **Image2, int width,
int height, int th) {
    int centerX1, centerY1, centerX2, centerY2;
    FindCenter(Image1, width, height, &centerX1, &centerY1, th);
    FindCenter(Image2, width, height, &centerX2, &centerY2, th);

    if (centerX1 != -1 && centerY1 != -1 && centerX2 != -1 && centerY2 != -1)
    {
        int deltaX = centerX2 - centerX1;
        int deltaY = centerY2 - centerY1;

        if (deltaX > 0) {
            printf("Di chuyển sang phải %d pixel.\n", abs(deltaX));
        } else if (deltaX < 0) {
            printf("Di chuyển sang trái %d pixel.\n", abs(deltaX));
        }
        if (deltaY > 0) {
            printf("Di chuyển xuống dưới %d pixel.\n", abs(deltaY));
        } else if (deltaY < 0) {
            printf("Di chuyển lên trên %d pixel.\n", abs(deltaY));
        }
        if (deltaX == 0 && deltaY == 0) {
            printf("Không di chuyển.\n");
        }
    }
}
```

- Hàm main, khai báo các giá trị kích thước ảnh, giá trị threshold. Sau đó thực hiện đọc ảnh màu RGB, chuyển sang GrayScale và lưu ảnh, thực hiện Edge Detection ra 2 ảnh gradient, sau đó phân tách vật thể và nền với Histogram, cuối cùng là xác định phương hướng di chuyển và số pixel di chuyển theo từng phương bằng cách đọc ảnh của 2 vật thể và xác định khoảng cách di chuyển của tâm 2 ảnh.

```
int main() {
    int width, height, th = 190;
    unsigned char *RGB1 = ReadBMPimg("G:\\ImgProcess\\BTVN6\\frame1.bmp",
    &width, &height);
    unsigned char *RGB2 = ReadBMPimg("G:\\ImgProcess\\BTVN6\\frame2.bmp",
    &width, &height);
    unsigned char **grayImage1 = MatrixMemory(width, height);
    unsigned char **grayImage2 = MatrixMemory(width, height);
```

```
    RGB2Gray(RGB1, grayImage1, width, height);
    RGB2Gray(RGB2, grayImage2, width, height);
    WriteGrayBMP("G:\\ImgProcess\\BTVN6\\Gray_frame1.bmp", grayImage1, width,
height);
    WriteGrayBMP("G:\\ImgProcess\\BTVN6\\Gray_frame2.bmp", grayImage2, width,
height);

    unsigned char **gradient1 = MatrixMemory(width, height);
    unsigned char **gradient2 = MatrixMemory(width, height);
    SobelKernel(grayImage1, width, height, gradient1);
    SobelKernel(grayImage2, width, height, gradient2);
    WriteGrayBMP("G:\\ImgProcess\\BTVN6\\gradient1.bmp", gradient1, width,
height);
    WriteGrayBMP("G:\\ImgProcess\\BTVN6\\gradient2.bmp", gradient2, width,
height);

    unsigned char **Object1 = MatrixMemory(width, height);
    unsigned char **Object2 = MatrixMemory(width, height);
    Histogram(gradient1, Object1, width, height, th);
    Histogram(gradient2, Object2, width, height, th);
    WriteGrayBMP("G:\\ImgProcess\\BTVN6\\Object1.bmp", Object1, width,
height);
    WriteGrayBMP("G:\\ImgProcess\\BTVN6\\Object2.bmp", Object2, width,
height);

    MotionDetect(Object1, Object2, width, height, th);

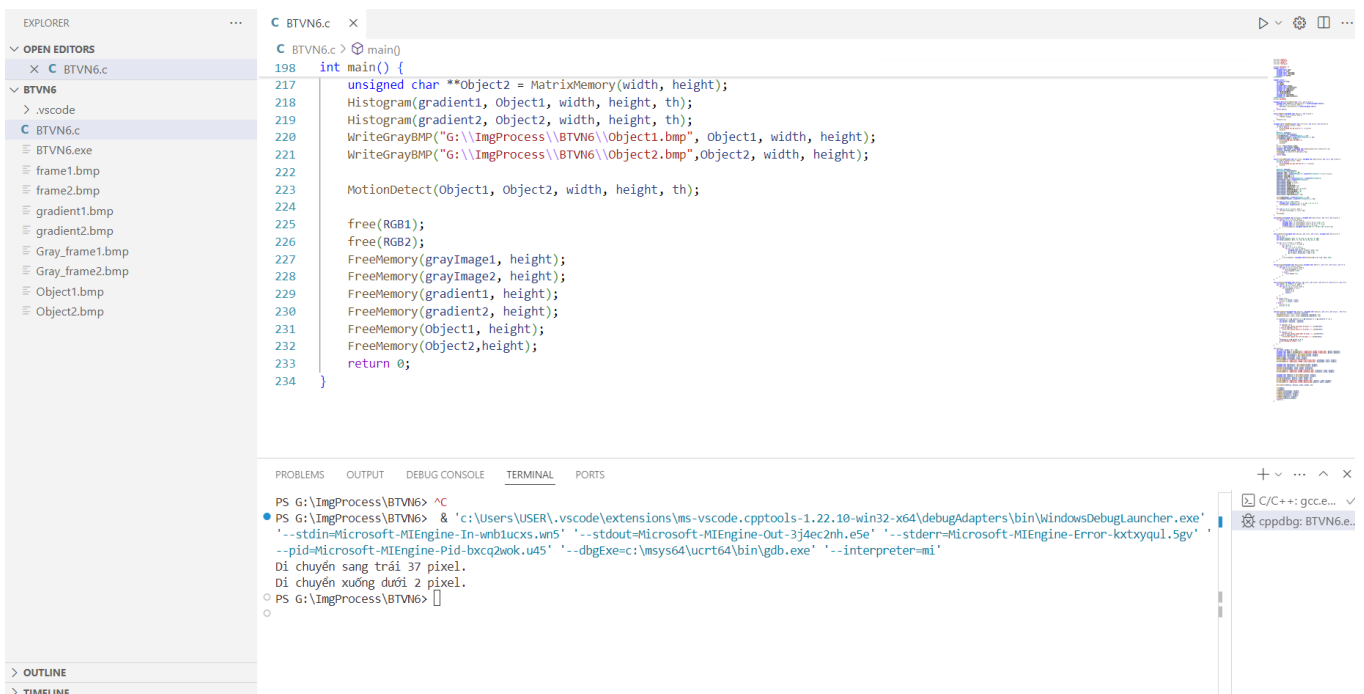
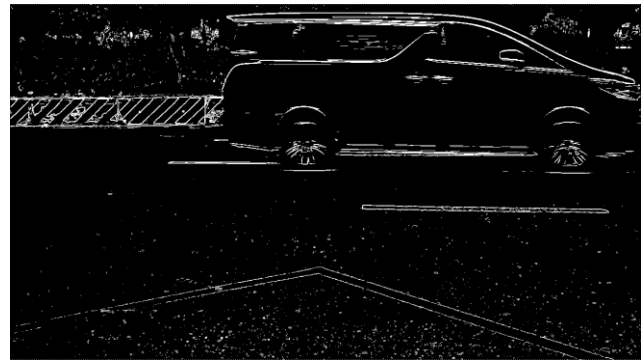
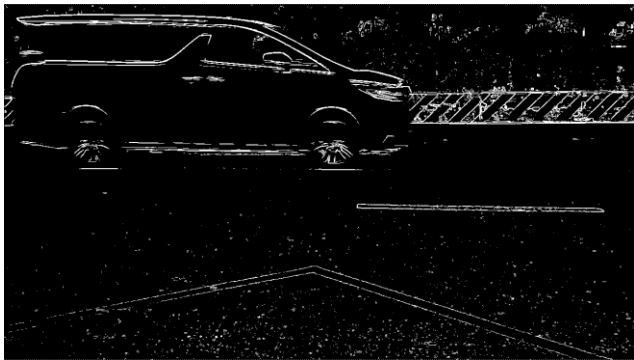
    free(RGB1);
    free(RGB2);
    FreeMemory(grayImage1, height);
    FreeMemory(grayImage2, height);
    FreeMemory(gradient1, height);
    FreeMemory(gradient2, height);
    FreeMemory(Object1, height);
    FreeMemory(Object2, height);
    return 0;
}
```

2. KẾT QUẢ CHẠY THỬ TRÊN VS CODE

- Kết quả Edge Detection



- Kết quả phân tách vật thể và nền



```
EXPLORER
  OPEN EDITORS
    C BTVN6.c
  BTVN6
    .vscode
    C BTVN6.c
    BTVN6.exe
    frame1.bmp
    frame2.bmp
    gradient1.bmp
    gradient2.bmp
    Gray_frame1.bmp
    Gray_frame2.bmp
    Object1.bmp
    Object2.bmp

C BTVN6.c
198 int main() {
199     unsigned char **Object2 = MatrixMemory(width, height);
200     Histogram(gradient1, Object1, width, height, th);
201     Histogram(gradient2, Object2, width, height, th);
202     WriteGrayBMP("G:\\ImgProcess\\BTVN6\\Object1.bmp", Object1, width, height);
203     WriteGrayBMP("G:\\ImgProcess\\BTVN6\\Object2.bmp", Object2, width, height);
204
205     MotionDetect(Object1, Object2, width, height, th);
206
207     free(RGB1);
208     free(RGB2);
209     FreeMemory(grayImage1, height);
210     FreeMemory(grayImage2, height);
211     FreeMemory(gradient1, height);
212     FreeMemory(gradient2, height);
213     FreeMemory(Object1, height);
214     FreeMemory(Object2, height);
215     return 0;
216 }
```

```
PS G:\ImgProcess\BTVN6> ^C
PS G:\ImgProcess\BTVN6> & 'c:\Users\USER\.vscode\extensions\ms-vscode.cpptools-1.22.10-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe'
'--stdin=Microsoft-MIEngine-In-wnblucxs.wn5' '--stdout=Microsoft-MIEngine-Out-3j4ec2nh.e5e' '--stderr=Microsoft-MIEngine-Error-kctxyql.5gv'
--pid=Microsoft-MIEngine-Pid-bxcq2wok.u45' '--dbgExe=c:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Di chuyển sang trái 37 pixel.
Di chuyển xuống dưới 2 pixel.
PS G:\ImgProcess\BTVN6>
```


Link folder chứa source code và các tài nguyên hình ảnh: https://studenthcmusedu-my.sharepoint.com/:f:/g/personal/20200331_student_hcmus_edu_vn/EiSPfp7yyKxKgJb1d3kJaQ4BDZ9NCEQQ4YZ6q13HIZ6tzQ?e=QXs877

--HẾT--