c\hough_multi.c

```c
// Objectif : utiliser du multithreading pour améliorer la rapidité
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include <pthread.h>

int PRECISION = 10;
char* filepath = "./in.txt";

int COLONNES;
int LIGNES;
int DISTANCE;
int* IMAGE;
int* ESPACE;

// taille du fichier. update global variables
int* taille(char* filepath){
    FILE* ptr;
    char ch;
    ptr = fopen(filepath, "r");
    if (NULL == ptr) {
        printf("file can't be opened \n");
    }
    int lignes = 0;
    int colonnes = 1;
    int end = 1;
    while (!feof(ptr)) {
        ch = fgetc(ptr);
        if (ch == '\n'){
            lignes ++;
            end = 0;
        }
        else if(ch == ',' && end) {
            colonnes ++;
        }
```

```
    }

    LIGNES = lignes;
    COLONNES = colonnes;
    DISTANCE = (int)round(sqrt(COLONNES *COLONNES + LIGNES *LIGNES));
}
//retourne la matrice
int* read_img(char* filepath){
    int* matrice = malloc((COLONNES * LIGNES) * sizeof(int));

    FILE* ptr;
    char ch;
    ptr = fopen(filepath, "r");

    if (NULL == ptr) {
        printf("file can't be opened \n");
    }
    int c = 0;
    while (!feof(ptr)) {
        ch = fgetc(ptr);
        if (feof(ptr)){
            fclose(ptr);
            IMAGE = matrice;
            return NULL;
        }
        if (ch == '\n'){
            c++;
        }
        else if(ch == ',') {
            c++;
        }
        else if (ch == ' '){}
        else {
            //printf("%c\n", ch);
            matrice[c] *= 10;

            matrice[c] += (int)ch - 48;
        }
    }

}
```

```
79
80
81  double rho(int x, int y, double theta){
82      return(x * cos(theta) + y * sin(theta));
83  }
84
85
86  void* espace_hough(void* arg){
87      int thread = *(int*)arg; // numero du thread equivalent a la zone de degré qui va être parcourue
88      int v_rho;
89      double theta_rad;
90      for (int y = 0; y < LIGNES; y++){
91          for (int x = 0; x < COLONNES; x++){
92              if (IMAGE[y*COLONNES + x] > 250){
93                  for (int theta_deg = 360 * thread ; theta_deg < 360 * (thread+1) ;theta_deg ++){
94                     theta_rad = theta_deg /(180.0 * PRECISION) * M_PI;
95                     v_rho = rho(x, y, theta_rad); // determine R
96                     if (v_rho > 0){
97                          ESPACE[v_rho * 360 * PRECISION + theta_deg] += 1;
98                     }
99                  }
100             }
101         }
102     }
103     return(NULL);
104 }
105
106
107
108 void save_file(int* mat, int PRECISION){
109     FILE *ptr;
110     ptr = fopen("./out.txt","w");
111
112     for (int j = 0; j < DISTANCE; j++){
113         for (int i = 0; i<360 * PRECISION; i++){
114             fprintf(ptr, "%d", mat[j*360 * PRECISION + i]);
115             if (i != 360 * PRECISION - 1){
116                 fprintf(ptr, "%s",",");
117             }
118         }
119         fprintf(ptr, "%s","\n");
```

```
    }

    fclose(ptr);

}


int main(int argc, char *argv[])
{
    if (argc == 2){
        PRECISION = strtol(argv[1], NULL, 10);
    }
    taille(filepath);

    printf("Precision : %d\nLignes : %d\nColonnes : %d\nnombre de pixels : %d\nDistance : %d\n", PRECISION, LIGNES, COLONNES,(LIGNES)*(COLONNES), DISTANCE);
    read_img(filepath);

    int* mat_hough = malloc(360 * PRECISION * DISTANCE * sizeof(int));
    for (int i = 0; i<360 * DISTANCE * PRECISION; i++){
        mat_hough[i] = 0;
    }
    ESPACE = mat_hough;

    pthread_t* t = (pthread_t*)malloc(sizeof(pthread_t)*(PRECISION));
    int* tab_int = malloc(PRECISION * sizeof(int));
    for(int i = 0; i < PRECISION; i++){
        tab_int[i] = i;
    }

    for (int i=0; i < PRECISION; i++) {
        pthread_create(&t[i], NULL, espace_hough, (void*)&tab_int[i]);
    }

    for (int i=0; i < PRECISION; i++) {
        pthread_join(t[i], NULL);
    }

    save_file(ESPACE, PRECISION);

    return 0;
```

```
}


```