HMIN318M: Imagerie médicale et 3D - informatique

Rendu : TP Lecture, stockage d'images 3D et visualisation volumique

Tianning MA

M2 IMAGINA

17/09/2020

Table de matière

1. Introduction

Ce compte rendu est dédié au TP imagerie 3D - informatique. Le programme écrit en C/C++ en utilisant les bibliothques classiques (stdio.h, stdlib.h, math.h et iostream pour flux d'entrée et sortie).

J'ai réussi à réaliser la question 1 lecture et stockage d'image 3D, et une partie de question 2 volume rendering (MIP, AIP, MinIP).

Malheureusement, je ne suis pas très à l'aise à la manipulation du fichier en C, je n'ai pas trop réussi à finaliser l'écriture du fichier image brute. (la difficulté est surtout l'écriture du fichier 2 octets par 2octets).

2. Rendu et explication des exercices

A. Lecture et stockage d'images 3D

a. Lecture et stockage en mémoire

```
void readImage(char filename[], int dimX, int dimY, int dimZ){
    FILE *image;
    int taille_image = dimX * dimY * dimZ;
    unsigned char buffer1 = 0; unsigned char buffer2 = 0;

    if( (image = fopen(filename, "rb")) == NULL){
        std::cout<<"Erreur d'ouverture de l'image "<<filename<<std::endl;
        exit(EXIT_FAILURE);
    }
    else{
        for(size_t i = 0; i< taille_image; i++){
            if(fread(&buffer1, 1, 1, image) <=0 || fread(&buffer2, 1, 1, image)<=0){
                std::cout<<"Erreur en lecture de l'image"<<std::endl;
                exit(EXIT_FAILURE);
            }
            else{
                unsigned short val = buffer1 * 256 + buffer2;
                if(val > max) max = val;
                if(val < min) min = val;
                imageInputbuffer[i] = val;
            }
        }
        std::cout<<"Lecture de l'image réussi."<<std::endl;
        fclose(image);
    }
}
```

Ici j'ai crée en mémoire un buffer « imageInputbuffer » pour stocker toutes les données de voxels. En itérant sur chaque voxel, je compare la valeur courante et min / max, pour récupérer la valeur maximum et la valeur minimum de cette image 3D. Pour la lecture du fichier, je lis un octet puis un autre octet (ensemble qui formera la vraie valeur de ce pixel). J'ai fait attention de faire décaler le buffer1 vers le poid fort pour calculer la vraie valeurs

b. La fonction getValue(i,j,k) renvoie la valeur du voxel

```
float getValue(int i, int j, int k){
    return imageInputbuffer[k * (height * width) + width * (height-j-1) + i];
    // couche k - ligne j (sens inversé) - colonne i
}
```

La fonction getValue est simplement pour but de retourner une valeur de voxel. Ici, j'ai fait attention le sens de y. (problème de balayage)

c. Affichage de la valeur minimale et maximale des voxels de l'image & affichage de l'intensité d'un voxel de coordonnées rentrées par l'utilisateur

```cpp
unsigned int choix_image = 0; char filename[250]("");
std::cout<<"*****************Lecture et stockage de l'image***************"<<std::endl;
std::cout<<"Veuillez choisir l'image à lire : "<<std::endl;
std::cout<<"1 - t1-head.256x256x129.1.5x1.5x1.5.img"<<std::endl;
std::cout<<"2 - INCISIX.512x512x166.0.3613281x0.3613281x0.5.img"<<std::endl;
std::cout<<"3 - orange.256x256x64.0.3906x0.3906x1.0.img"<<std::endl;
std::cin>>choix_image;
std::cout<<"-- Lecture de l'image brute : ";
if(choix_image == 1){
    char filename[250]("t1-head.256x256x129.1.5x1.5x1.5.img"); std::cout<<filename<<std::endl;
    height = 256; width = 256; couche = 129;
    int taille_image = width * height * couche;
    allocation_tableau(imageInputbuffer, unsigned short, taille_image);
    readImage(filename, 256, 256, 129);
}
else if(choix_image == 2){
    char filename[250]("INCISIX.512x512x166.0.3613281x0.3613281x0.5.img");std::cout<<filename<<std::endl;
    height = 512; width = 512; couche = 166;
    int taille_image = width * height * couche;
    allocation_tableau(imageInputbuffer, unsigned short, taille_image);
    readImage(filename, 512, 512, 166);
}
else if(choix_image == 3){
    char filename[250]("orange.256x256x64.0.3906x0.3906x1.0.img");std::cout<<filename<<std::endl;
    height = 256; width = 256; couche = 64;
    int taille_image = width * height * couche;
    allocation_tableau(imageInputbuffer, unsigned short, taille_image);
    readImage(filename, 256, 256, 64);
}
else{
    std::cout<<"Choix invalid. Exit";
    exit(EXIT_FAILURE);
}


std::cout<<"-- Information sur l'image : "<<std::endl;
std::cout<< "\tmin : " << min << std::endl;
std::cout<< "\tmax : " << max << std::endl<<std::endl;

std::cout<<"*****************Recherche d'intensité du voxel donnée***************"<<std::endl;
int x(0), y(0), z(0);
std::cout<<"Veuillez entrez les coordonées du voxel : (x, y, z)"<<std::endl;
std::cin>>x>>y>>z;
std::cout<<"Voxel ("<<x<<","<<y<<","<<z<<") : "<<getValue(x,y,z)<<std::endl;
```

d. Testes

Test sur image orange : min = 0, max = 228, I(128,128,32) = 15

```
*****************Lecture et stockage de l'image***************
Veuillez choisir l'image à lire :
1 - t1-head.256x256x129.1.5x1.5x1.5.img
2 - INCISIX.512x512x166.0.3613281x0.3613281x0.5.img
3 - orange.256x256x64.0.3906x0.3906x1.0.img
3
-- Lecture de l'image brute : orange.256x256x64.0.3906x0.3906x1.0.img
Lecture de l'image réussi.
-- Information sur l'image :
        min : 0
        max : 228

*****************Recherche d'intensité du voxel donnée***************
Veuillez entrez les coordonées du voxel : (x, y, z)
128 128 32
Voxel (128,128,32) : 15
```

Test sur INCISIX : min = 0, max = 4095, I(184,343,83) = 2567

```
******************Lecture et stockage de l'image***************
Veuillez choisir l'image à lire :
1 - t1-head.256x256x129.1.5x1.5x1.5.img
2 - INCISIX.512x512x166.0.3613281x0.3613281x0.5.img
3 - orange.256x256x64.0.3906x0.3906x1.0.img
2
-- Lecture de l'image brute : INCISIX.512x512x166.0.3613281x0.3613281x0.5.img
Lecture de l'image réussi.
-- Information sur l'image :
        min : 0
        max : 4095

******************Recherche d'intensité du voxel donnée***************
Veuillez entrez les coordonées du voxel : (x, y, z)
184 343 83
Voxel (184,343,83) : 2567
```

Test sur T1-head : min = 0, max = 885, I(158,143,64) = 300

```
******************Lecture et stockage de l'image***************
Veuillez choisir l'image à lire :
1 - t1-head.256x256x129.1.5x1.5x1.5.img
2 - INCISIX.512x512x166.0.3613281x0.3613281x0.5.img
3 - orange.256x256x64.0.3906x0.3906x1.0.img
1
-- Lecture de l'image brute : t1-head.256x256x129.1.5x1.5x1.5.img
Lecture de l'image réussi.
-- Information sur l'image :
        min : 0
        max : 885

******************Recherche d'intensité du voxel donnée***************
Veuillez entrez les coordonées du voxel : (x, y, z)
158 143 64
Voxel (158,143,64) : 300
```

B. Volume Rendering (MIP, AIP, MinIP suivant les directions axiales x, y ,z) et Ecriture du fichier image brute

a. Volume Rendering

visuAxis 1 = x, 2 = y, 3 = z visuMode 1 = MIP, 2 = AIP, 3 = MinIP

```
void VolumeRendering(char filename[], int dimX, int dimY, int dimZ,
    char resultFile[], unsigned int visuAxis = 1, unsigned int visuMode = 1){

    unsigned short *imageOutputbuffer;

    //visuAxis 1=x, 2=y, 3=z
    allocation_tableau(imageOutputbuffer, unsigned short, dimZ * dimY);
    if(visuAxis == 1){
        for(size_t z = 0; z < dimZ; z++){
            for(size_t y = 0; y < dimY; y++){
                unsigned int max_x = 0;
                unsigned int min_x = 65535;
                int accum_x = 0;
                for(size_t x = 0; x < dimX; x++){
                    if(visuMode == 1){
                        if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] > max_x)
                            max_x = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                    }
                    if(visuMode == 2){
                        accum_x += imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                    }
                    if(visuMode == 3){
                        if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] < min_x)
                            min_x = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                    }
                }
if(visuAxis == 2){ // axe y
    allocation_tableau(imageOutputbuffer, unsigned short, dimX * dimZ);
    for(size_t z = 0; z < dimZ; z++){
        for(size_t x = 0; x < dimX; x++){
            unsigned int max_y = 0;
            unsigned int min_y = 65535;
            int accum_y = 0;
            for(size_t y = 0; y < dimY; y++){
                if(visuMode == 1){
                    if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] > max_y)
                        max_y = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                }
                if(visuMode == 2){
                    accum_y += imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                }
                if(visuMode == 3){
                    if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] < min_y)
                        min_y = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                }

            }
            if(visuMode == 1) imageOutputbuffer[dimX * x + z] = max_y;
            if(visuMode == 2) imageOutputbuffer[dimX * x + z] = accum_y / dimY;
            if(visuMode == 3) imageOutputbuffer[dimX * x + z] = min_y;
        }
    }
}
```

```
if(visuAxis == 3){ // axe z
    allocation_tableau(imageOutputbuffer, unsigned short, dimX * dimY);
    for(size_t x = 0; x < dimX; x++){
        for(size_t y = 0; y < dimY; y++){
            unsigned int max_z = 0;
            unsigned int min_z = 65535;
            int accum_z = 0;
            for(size_t z = 0; z < dimZ; z++){
                if(visuMode == 1){
                    if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] > max_z)
                        max_z = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                }
                if(visuMode == 2){
                        accum_z += imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                }
                if(visuMode == 3){
                    if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] < min_z)
                            min_z = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                }
            }
            if(visuMode == 1) imageOutputbuffer[dimX * y + x] = max_z;
            if(visuMode == 2) imageOutputbuffer[dimX * y + x] = accum_z / dimZ;
            if(visuMode == 3) imageOutputbuffer[dimX * y + x] = min_z;
        }
    }
}

if(visuMode!=1 && visuMode!=2 && visuMode!=3) {
    std::cout<<"Mode invalid. Exit"<<std::endl; exit(EXIT_FAILURE);
}
```

Les photos ci-dessus sont pour la partie Volume Rendering qui consiste à calculer les pixels des images en MIP / AIP / MinIP. L'idée est de parcourir une axe donné et déterminer une valeur par pixel selon les critères. Ensuite, je sauvegarde ces données dans un buffer pour image de sortie.

b. Ecriture du fichier image brute

```
//Ecriture de fichier

FILE *f_image;
unsigned char buffer1 = 0; unsigned char buffer2 = 0;
if((f_image = fopen(resultFile, "wb")) == NULL){
    std::cout<<"Erreur en ecriture sur l'image "<<resultFile<<std::endl;
    exit(EXIT_FAILURE);
}
else{
    if((fwrite(imageOutputbuffer, 1, 1, f_image)) <=0 || (fwrite(imageOutputbuffer, 1, 1, f_image) <=0)){
        std::cout<<"Erreur d'écriture de l'image "<<resultFile<<std::endl;
        exit(EXIT_FAILURE);
    }
    fclose(f_image);
}
```

(Je ne suis pas sûr de cette partie.)

Défi : qu'est-ce que whatisit ?

Selon la forme et la proportion de l'air et tissu, je pense c'est peut-être une méduse ?

Annexe : Code source

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

#define allocation_tableau(nom, type, nombre) \
if( (nom = (type*) calloc (nombre, sizeof(type) ) ) == NULL ) \
{\
 printf("\n Allocation dynamique impossible pour un pointeur-tableau \n");\
 exit(1);\
}

/*****Variables globales*****/
unsigned short *imageInputbuffer;
unsigned short *imageOutputbuffer;
int width = 0;                          // x - colonne
int height = 0;          // y - ligne
int couche = 0;          // z
unsigned short min = 65535; int index_min = 0;
unsigned short max = 0;             int index_max = 0;



void readImage(char filename[], int dimX, int dimY, int dimZ){
            FILE *image;
            int taille_image = dimX * dimY * dimZ;
            unsigned char buffer1 = 0; unsigned char buffer2 = 0;

            if( (image = fopen(filename, "rb")) == NULL){
                        std::cout<<"Erreur d'ouverture de l'image "<<filename<<std::endl;
                        exit(EXIT_FAILURE);
            }
            else{
                        for(size_t i = 0; i< taille_image; i++){
                                    if(fread(&buffer1, 1, 1, image) <=0 || fread(&buffer2, 1, 1, image)<=0){
                                                std::cout<<"Erreur en lecture de l'image"<<std::endl;
                                                exit(EXIT_FAILURE);
                                    }
                                    else{
                                                unsigned short val = buffer1 * 256 + buffer2;
                                                if(val > max) max = val;
                                                if(val < min) min = val;
                                                imageInputbuffer[i] = val;
```

```
                }
            }
            std::cout<<"Lecture de l'image réussi."<<std::endl;
            fclose(image);
        }
}

float getValue(int i, int j, int k){
    return imageInputbuffer[k * (height * width) + width * (height-j-1) + i]; // couche k - ligne j (sens inversé) - colonne i
}


void VolumeRendering(char filename[], int dimX, int dimY, int dimZ,
    char resultFile[], unsigned int visuAxis = 1, unsigned int visuMode = 1){

    unsigned short *imageOutputbuffer;

    //visuAxis 1=x, 2=y, 3=z
    allocation_tableau(imageOutputbuffer, unsigned short, dimZ * dimY);
    if(visuAxis == 1){
        for(size_t z = 0; z < dimZ; z++){
            for(size_t y = 0; y < dimY; y++){
                unsigned int max_x = 0;
                unsigned int min_x = 65535;
                int accum_x = 0;
                for(size_t x = 0; x < dimX; x++){
                    if(visuMode == 1){
                        if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] > max_x)
                            max_x = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                    }
                    if(visuMode == 2){
                        accum_x += imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                    }
                    if(visuMode == 3){
                        if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] < min_x)
                            min_x = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                    }
                }
                if(visuMode == 1) imageOutputbuffer[dimZ * y + z] = max_x;
                if(visuMode == 2) imageOutputbuffer[dimZ * y + z] = accum_x / dimX;
                if(visuMode == 3) imageOutputbuffer[dimZ * y + z] = min_x;
            }
        }
```

```
}
if(visuAxis == 2){ // axe y
		allocation_tableau(imageOutputbuffer, unsigned short, dimX * dimZ);
		for(size_t z = 0; z < dimZ; z++){
			for(size_t x = 0; x < dimX; x++){
				unsigned int max_y = 0;
				unsigned int min_y = 65535;
				int accum_y = 0;
				for(size_t y = 0; y < dimY; y++){
					if(visuMode == 1){
						if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] > max_y)
							max_y = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
					}
					if(visuMode == 2){
						accum_y += imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
					}
					if(visuMode == 3){
						if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] < min_y)
							min_y = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
					}

				}
				if(visuMode == 1) imageOutputbuffer[dimX * x + z] = max_y;
				if(visuMode == 2) imageOutputbuffer[dimX * x + z] = accum_y / dimY;
				if(visuMode == 3) imageOutputbuffer[dimX * x + z] = min_y;
			}
		}
}
if(visuAxis == 3){ // axe z
		allocation_tableau(imageOutputbuffer, unsigned short, dimX * dimY);
		for(size_t x = 0; x < dimX; x++){
			for(size_t y = 0; y < dimY; y++){
				unsigned int max_z = 0;
				unsigned int min_z = 65535;
				int accum_z = 0;
				for(size_t z = 0; z < dimZ; z++){
					if(visuMode == 1){
						if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] > max_z)
							max_z = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
					}
					if(visuMode == 2){
						accum_z += imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
					}
```

```
                                        if(visuMode == 3){
                                                if(imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x] < min_z)
                                                        min_z = imageInputbuffer[z * (dimY * dimX) + dimX * (dimY-y-1) + x];
                                        }
                                }
                                if(visuMode == 1) imageOutputbuffer[dimX * y + x] = max_z;
                                if(visuMode == 2) imageOutputbuffer[dimX * y + x] = accum_z / dimZ;
                                if(visuMode == 3) imageOutputbuffer[dimX * y + x] = min_z;
                        }
                }
        }

        if(visuMode!=1 && visuMode!=2 && visuMode!=3) {
                std::cout<<"Mode invalid. Exit"<<std::endl; exit(EXIT_FAILURE);
        }

        //Ecriture de fichier

        FILE *f_image;
        unsigned char buffer1 = 0; unsigned char buffer2 = 0;
        if((f_image = fopen(resultFile, "wb")) == NULL){
                std::cout<<"Erreur en ecriture sur l'image "<<resultFile<<std::endl;
                exit(EXIT_FAILURE);
        }
        else{
                if((fwrite(imageOutputbuffer, 1, 1, f_image)) <=0 || (fwrite(imageOutputbuffer, 1, 1, f_image) <=0)){
            std::cout<<"Erreur d'écriture de l'image "<<resultFile<<std::endl;
            exit(EXIT_FAILURE);
                }
                fclose(f_image);
        }


}


int main(int argc, char const *argv[])
{
        unsigned int choix_image = 0; char filename[250]("");
        std::cout<<"******************Lecture et stockage de l'image*****************"<<std::endl;
        std::cout<<"Veuillez choisir l'image à lire : "<<std::endl;
        std::cout<<"1 - t1-head.256x256x129.1.5x1.5x1.5.img"<<std::endl;
        std::cout<<"2 - INCISIX.512x512x166.0.3613281x0.3613281x0.5.img"<<std::endl;
```

```
	std::cout<<"3 - orange.256x256x64.0.3906x0.3906x1.0.img"<<std::endl;
	std::cin>>choix_image;
	std::cout<<"-- Lecture de l'image brute : ";
	if(choix_image == 1){
		char filename[250]("t1-head.256x256x129.1.5x1.5x1.5.img"); std::cout<<filename<<std::endl;
		height = 256; width = 256; couche = 129;
		int taille_image = width * height * couche;
		allocation_tableau(imageInputbuffer, unsigned short, taille_image);
		readImage(filename, 256, 256, 129);
	}
	else if(choix_image == 2){
		char filename[250]("INCISIX.512x512x166.0.3613281x0.3613281x0.5.img");std::cout<<filename<<std::endl;
		height = 512; width = 512; couche = 166;
		int taille_image = width * height * couche;
		allocation_tableau(imageInputbuffer, unsigned short, taille_image);
		readImage(filename, 512, 512, 166);
	}
	else if(choix_image == 3){
		char filename[250]("orange.256x256x64.0.3906x0.3906x1.0.img");std::cout<<filename<<std::endl;
		height = 256; width = 256; couche = 64;
		int taille_image = width * height * couche;
		allocation_tableau(imageInputbuffer, unsigned short, taille_image);
		readImage(filename, 256, 256, 64);
	}
	else{
		std::cout<<"Choix invalid. Exit";
		exit(EXIT_FAILURE);
	}
	std::cout<<"-- Information sur l'image : "<<std::endl;
	std::cout<< "\tmin : " << min << std::endl;
	std::cout<< "\tmax : " << max << std::endl<<std::endl;

	std::cout<<"******************Recherche d'intensité du voxel donnée****************"<<std::endl;
	int x(0), y(0), z(0);
	std::cout<<"Veuillez entrez les coordonées du voxel : (x, y, z)"<<std::endl;
	std::cin>>x>>y>>z;
	std::cout<<"Voxel ("<<x<<","<<y<<","<<z<<") : "<<getValue(x,y,z)<<std::endl;

	char output[250]("output.img");
	VolumeRendering(filename, width, height, couche, output, 1,1);
	return 0;
}
```