Analyse et traitement d'images

Compte-rendu TP Imagerie 3D n°1

Lecture, stockage d'images 3D et visualisation volumique

Louis Jean
Master 1 IMAGINE
Université de Montpellier
N° étudiant : 21914083

5 avril 2024

Table des matières

1 Introduction

Ce travail propose une introduction au traitement d'images 3D appliqué au domaine de l'imagerie médicale. Ce TP est structuré autour de deux axes principaux : d'une part, la lecture et le stockage d'images 3D à partir de données brutes ; d'autre part, l'exploration de techniques de visualisation volumique telles que le Maximum Intensity Projection (MIP), l'Average Intensity Projection (AIP), et le Minimum Intensity Projection (MINIP), chacune offrant une perspective unique sur les données tridimensionnelles traitées.

2 Lecture et stockage d'images 3D au format brut

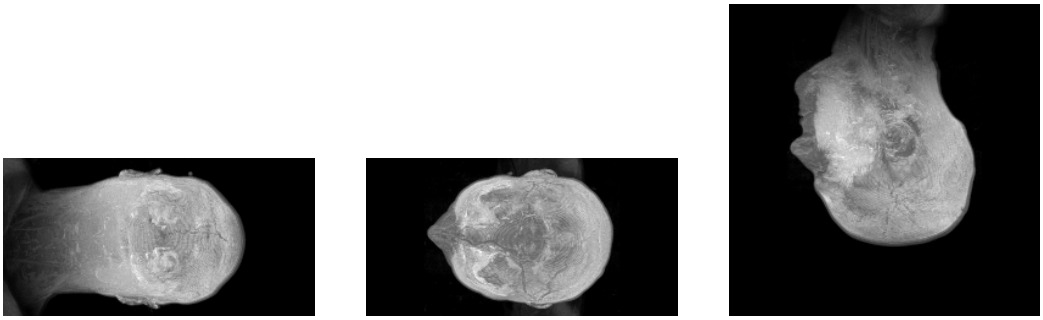
Pour lire et stocker une image, j'ai choisi de créer une structure comprenant un vecteur d'entiers (représentant les valeurs des voxels) et les dimensions de l'image sur chaque axe. À cela s'ajoutent des méthodes pour parcourir l'image et récupérer ou définir les valeurs des voxels, qui respectent le balayage indiqué dans le sujet. Pour charger une image, j'ai écrit une fonction qui lit le fichier voulu 2 octets par 2 octets, en faisant bien attention que l'on est dans le format Big Endian (voir slide 35 du cours). Après avoir testé mes fonctions sur les images fournies, j'obtiens bien les valeurs attendues pour la valeur minimale et maximale des voxels, aussi bien que pour les voxels prédéfinis dans le sujet.

3 Volume rendering

Pour le volume rendering, j'ai implémenté les algorithmes MIP, AIP, et MINIP. Il faut d'abord faire une projection sur l'axe considéré, puis :

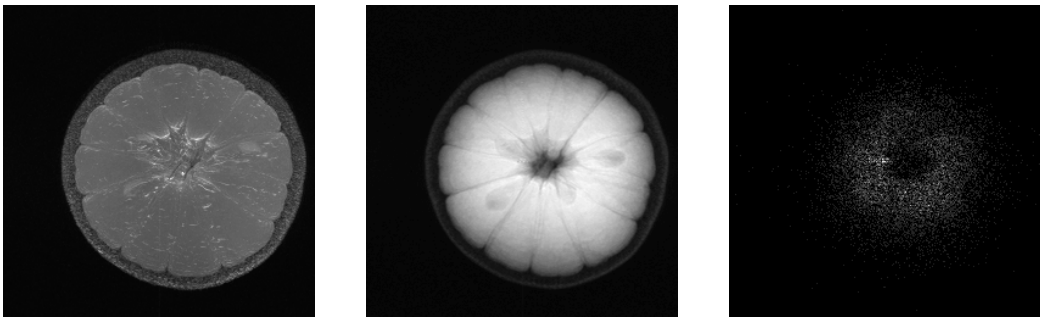
- **Pour MIP :** cette technique consiste à parcourir chaque colonne de voxels le long de l'axe de projection et à sélectionner la valeur maximale rencontrée. Cette valeur représente l'intensité du pixel dans l'image projetée. Le MIP est particulièrement utile pour mettre en évidence les structures les plus brillantes dans le volume, comme les os en imagerie médicale.
- **Pour AIP :** contrairement au MIP, l'AIP calcule la moyenne des intensités de tous les voxels le long de l'axe de projection pour chaque colonne.
- **Pour MINIP :** cette technique est l'opposée du MIP et sélectionne la valeur minimale de l'intensité des voxels pour chaque colonne le long de l'axe de projection. Le MINIP est particulièrement efficace pour visualiser les cavités ou les canaux dans le volume.

4 Rendu sur les images de test



(a) Axe x (b) Axe y (c) Axe z

FIGURE 1 – t1-head.img avec la méthode MIP sur plusieurs axes différents



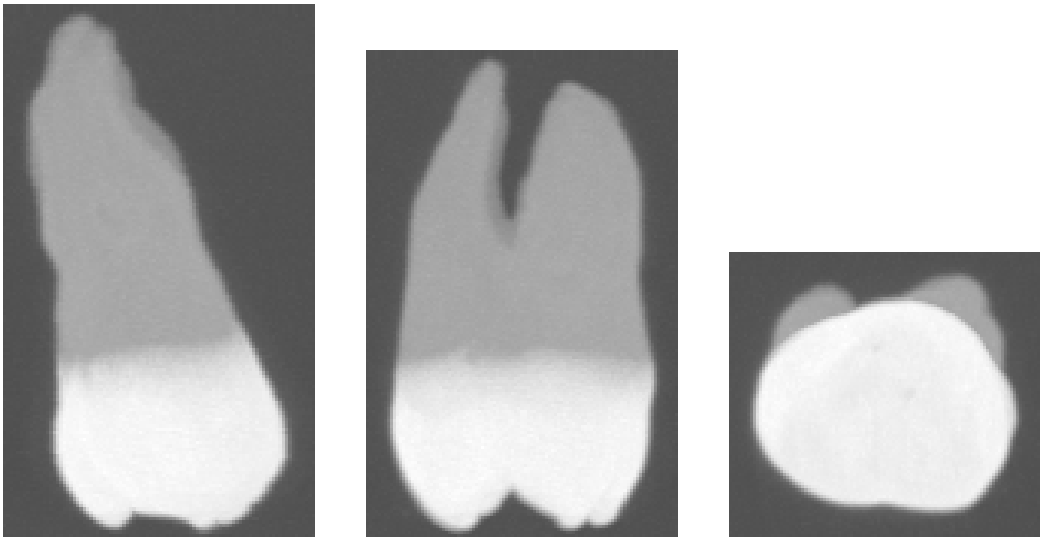
(a) MIP (b) AIP (c) MINIP

FIGURE 2 – orange.img sur l'axe z selon plusieurs méthodes

FIGURE 3 – incisix.img selon l'axe z avec la méthode MIP

5 Reconnaissance de l'image *whatisit*

On observe que *whatisit.img* semble être l'image d'une dent.



(a) Axe x (b) Axe y (c) Axe z

FIGURE 4 – whatisit.img selon plusieurs axes avec la méthode MIP

6 Code source

6.1 Image3D.hpp

```cpp
#include <iostream>
#include <vector>
#include <limits>
#include <algorithm>
#include <numeric>

struct Image3D {
    std::vector<unsigned short> data; // Stockage des voxels
    int dimX, dimY, dimZ; // Stockage des dimensions

    // Constructeur
    Image3D(int x, int y, int z) : dimX(x), dimY(y), dimZ(z)
    {
        // Il y a dimX * dimY * dimZ voxels dans l'image
        data.resize(dimX * dimY * dimZ);
    }

    // Méthode pour récupérer l'index d'un voxel
    int get_index(int i, int j, int k) const {
        return k * dimX * dimY + j * dimX + i; // Adapté pour le balayage de l'annexe C
        // k * dimX * dimY pour se déplacer de coupe en coupe
        // j * dimX pour se déplacer de ligne en ligne à l'intérieur d'une coupe
        // i pour se déplacer de colonne en colonne à l'intérieur d'une ligne
    }

    // Méthode pour récupérer la valeur d'un voxel
    unsigned short get_value(int i, int j, int k) const {
        return data[get_index(i,j,k)];
    }

    // Méthode pour définir la valeur d'un voxel
    void set_value(int i, int j, int k, unsigned short value)
    {
        data[get_index(i,j,k)] = value;
    }
};

// Fonction pour charger une image
void load_image(char* image_read, Image3D &image3D) {
    FILE* file = fopen(image_read, "r");
    if (!file) {
```

```
40          std::cout<<"Erreur lors de l'ouverture de l'image"<<
    std::endl;
41          exit(EXIT_FAILURE);
42      }
43      // On lit de k en i pour respecter le balayage
44      for(int k = 0; k < image3D.dimZ; ++k) {
45          for(int j = 0; j < image3D.dimY; ++j) {
46              for(int i = 0; i < image3D.dimX; ++i) {
47                  unsigned char bytes[2]; // Ici, les voxels
    sont stockés sur 2 octets
48                  if(fread(&bytes, sizeof(unsigned char), 2,
    file) != 2) { // Il faut indiquer à fread que l'on veut
    lire 2 octets
49                      std::cout<<"Erreur lors de la lecture de
    l'image"<<std::endl;
50                      exit(EXIT_FAILURE);
51                  }
52                  unsigned short current_voxel_read = 256 *
    bytes[0] + bytes[1]; // Comme dans l'exemple de la slide
    35 du cours (format Big Endian), si on fait 256 * bytes[1]
     + bytes[0] alors on trouve 58368 (erreur mentionnée au
    tableau)
53                  image3D.set_value(i,j,k,current_voxel_read);
54              }
55          }
56      }
57  }
58
59  // Fonction pour trouver la valeur minimale et maximale des
    voxels d'une image 3D
60  void find_min_max(Image3D &image3D, unsigned short &min,
    unsigned short &max) {
61      min = std::numeric_limits<unsigned short>::max();
62      max = std::numeric_limits<unsigned short>::min();
63      for(int i = 0; i < image3D.dimX; ++i) {
64          for(int j = 0; j < image3D.dimY; ++j) {
65              for(int k = 0; k < image3D.dimZ; ++k) {
66                  unsigned short current_voxel = image3D.
    get_value(i,j,k);
67                  if(current_voxel < min) min = current_voxel;
68                  if(current_voxel > max) max = current_voxel;
69              }
70          }
71      }
72  }
73
74  // Fonction pour écrire une image au format pgm (reprise des
    TPs avec M. Puech et adaptée pour traiter avec des
    unsigned short)
```

```
75 void ecrire_image_pgm(const char nom_image[], std::vector<
   unsigned short>& pt_image, int nb_lignes, int nb_colonnes)
    {
76    FILE *f_image;
77    int taille_image = nb_colonnes * nb_lignes;
78    std::vector<unsigned char> buffer(taille_image); // Buffer
    pour stocker les données converties
79
80    // Conversion
81    unsigned short max_val = *std::max_element(pt_image.begin
   (), pt_image.end());
82    for(int i = 0; i < taille_image; ++i) {
83        // Normalisation
84        buffer[i] = static_cast<unsigned char>(255.0 *
   pt_image[i] / max_val);
85    }
86
87    if((f_image = fopen(nom_image, "wb")) == NULL) {
88       printf("\nPas d'acces en ecriture sur l'image %s \n",
   nom_image);
89       exit(EXIT_FAILURE);
90    } else {
91       fprintf(f_image, "P5\n%d %d\n255\n", nb_colonnes,
   nb_lignes);
92       fwrite(buffer.data(), sizeof(unsigned char),
   taille_image, f_image);
93       fclose(f_image);
94    }
95 }
96
97 // Fonction pour générer la projection choisie et écrire l'
   image correspondante
98 void generate_projection(Image3D &image3D, std::vector<
   unsigned short> &projection, int axis, int mode, const
   char* image_write) {
99    int width, height, depth;
100   switch(axis) {
101       case 1: // Axe x
102           width = image3D.dimY;
103           height = image3D.dimZ;
104           depth = image3D.dimX;
105           break;
106       case 2: // Axe y
107           width = image3D.dimX;
108           height = image3D.dimZ;
109           depth = image3D.dimY;
110           break;
111       case 3: // Axe z
112           width = image3D.dimX;
```

```
            height = image3D.dimY;
            depth = image3D.dimZ;
            break;
        default:
            std::cout<<"Axe invalide"<<std::endl;
            exit(EXIT_FAILURE);
            return;
    }

    projection.resize(width * height, (mode == 3) ? std::
    numeric_limits<unsigned short>::max() : 0); // Initialiser
     pour MIP/AIP avec 0, pour MINIP avec max

    for(int x = 0; x < width; ++x) {
        for(int y = 0; y < height; ++y) {
            std::vector<unsigned short> values(depth); //
    Stocke les valeurs le long de l'axe de profondeur
            for(int d = 0; d < depth; ++d) {
                int i, j, k;
                switch(axis) {
                    case 1: i = d; j = x; k = y; break; //
    Axe x
                    case 2: i = x; j = d; k = y; break; //
    Axe y
                    case 3: i = x; j = y; k = d; break; //
    Axe z
                }
                values[d] = image3D.get_value(i, j, k);
            }
            unsigned short projected_voxel;
            if(mode == 1) { // MIP
                projected_voxel = *std::max_element(values.
    begin(), values.end());
            } else if(mode == 2) { // AIP
                unsigned long sum = std::accumulate(values.
    begin(), values.end(), 0UL); // Trick pour calculer la
    somme
                projected_voxel = (unsigned short)(sum /
    values.size());
            } else if(mode == 3) { // MINIP
                projected_voxel = *std::min_element(values.
    begin(), values.end());
            }
            projection[y * width + x] = projected_voxel;
        }
    }
    ecrire_image_pgm(image_write, projection, height, width);
}
```

6.2 lecture_et_stockage.cpp

```cpp
#include "Image3D.hpp"

int main(int argc, char* argv[]) {

    if(argc != 5) {
        std::cout<<"Utilisation : "<<argv[0]<<" <nom_image_lue.img> <dimX> <dimY> <dimZ>"<<std::endl;
        exit(EXIT_FAILURE);
    }

    // On stocke le nom de l'image lue
    char image_read[256];
    sscanf(argv[1],"%s",image_read);

    // On stocke les dimensions nécessaires pour lire l'image
    int dimX, dimY, dimZ;
    dimX = atoi(argv[2]);
    dimY = atoi(argv[3]);
    dimZ = atoi(argv[4]);

    // On initialise la structure l'image 3D
    Image3D image3D(dimX, dimY, dimZ);

    // On charge l'image 3D dans notre structure
    load_image(image_read, image3D);

    // On demande de choisir un voxel pour ensuite afficher sa valeur
    int i, j, k;
    std::cout<<"Entrez les coordonnées du voxel (i j k) : ";
    std::cin>>i>>j>>k;
    std::cout<<"Intensité du voxel ["<<i<<","<<j<<","<<k<<"] : "<<image3D.get_value(i,j,k)<<std::endl;

    // On affiche la valeur minimale et maximale des voxels de l'image
    unsigned short min, max;
    find_min_max(image3D,min,max);
    std::cout<<"Valeur minimale : "<<min<<std::endl;
    std::cout<<"Valeur maximale : "<<max<<std::endl;

    return 0;

}
```

6.3 volume_rendering.cpp

```cpp
#include "Image3D.hpp"
#include <cstring>

int main(int argc, char* argv[]) {

    if(argc != 8) {
        std::cout<<"Utilisation : "<<argv[0]<<" <nom_image_lue.img> <dimX> <dimY> <dimZ> <nom_image_ecrite.pgm> <x|y|z> <mip|aip|minip>"<<std::endl;
        exit(EXIT_FAILURE);
    }

    // On stocke le nom de l'image lue
    char image_read[256];
    sscanf(argv[1],"%s",image_read);

    // On stocke le nom de l'image écrite
    char image_write[256];
    sscanf(argv[5],"%s",image_write);

    // On stocke les dimensions nécessaires pour lire l'image
    int dimX, dimY, dimZ;
    dimX = atoi(argv[2]);
    dimY = atoi(argv[3]);
    dimZ = atoi(argv[4]);

    // On stocke l'axe selon lequel projeter
    int axis = 1; // x par défaut
    char axis_argv;
    sscanf(argv[6],"%c",&axis_argv);
    if(axis_argv == 'y') axis = 2;
    if(axis_argv == 'z') axis = 3;

    // On stocke le mode de visualisation
    int mode = 1; // MIP par défaut
    char mode_argv[10];
    sscanf(argv[7],"%s",mode_argv);
    if(strcmp(mode_argv,"aip") == 0) mode = 2;
    if(strcmp(mode_argv,"minip") == 0) mode = 3;

    // On initialise la structure l'image 3D
    Image3D image3D(dimX, dimY, dimZ);

    // On charge l'image 3D dans notre structure
    load_image(image_read, image3D);

    // On effectue la projection (qui s'occupera aussi d'é
```

```
    crire l'image)
46      std::vector<unsigned short> projection;
47      generate_projection(image3D,projection,axis,mode,(const
        char*)image_write);
48
49      return 0;
50
51  }
```