Analyse et traitement d'images

Compte-rendu TP Imagerie 3D n°2

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

12 avril 2024

Table des matières

1 Introduction

Dans le cadre de ce travail pratique sur le traitement des images médicales, nous allons explorer l'extraction d'iso-surfaces à partir d'images 3D en utilisant une version simplifiée de l'algorithme du `Marching Cubes`. Cette méthode est cruciale pour visualiser et analyser les structures volumétriques complexes souvent rencontrées en imagerie médicale, comme les IRM ou les scans CT.

2 Extraction d'une iso-surface d'une image 3D sous la forme d'un maillage 3D

Pour extraire une iso-surface d'une image 3D, nous avons utilisé une version simplifiée de l'algorithme du `Marching Cubes`.

Algorithme 1 Extraction d'une iso-surface d'une image 3D

```
1: Fonction ExtractionIsoSurface(image_read, dimX, dimY, dimZ, sizeX, sizeY, sizeZ, seuil)
2:     image3D ← LoadImage(image_read, dimX, dimY, dimZ)
3:     for i ← 1 to dimX − 1 do
4:         for j ← 1 to dimY − 1 do
5:             for k ← 1 to dimZ − 1 do
6:                 current_value ← get_value(image3D, i, j, k)
7:                 if current_value > seuil then
8:                     vertices ← generate_voxel_vertices(i, j, k, sizeX, sizeY, sizeZ)
9:                     if get_value(image3D, i − 1, j, k) < seuil then
10:                        triangle1, triangle2 ← generate_triangles(vertices, "i-1")
11:                    end if
12:                    if get_value(image3D, i + 1, j, k) < seuil then
13:                        triangle1, triangle2 ← generate_triangles(vertices, "i+1")
14:                    end if
15:                    if get_value(image3D, i, j − 1, k) < seuil then
16:                        triangle1, triangle2 ← generate_triangles(vertices, "j-1")
17:                    end if
18:                    if get_value(image3D, i, j + 1, k) < seuil then
19:                        triangle1, triangle2 ← generate_triangles(vertices, "j+1")
20:                    end if
21:                    if get_value(image3D, i, j, k − 1) < seuil then
22:                        triangle1, triangle2 ← generate_triangles(vertices, "k-1")
23:                    end if
24:                    if get_value(image3D, i, j, k + 1) < seuil then
25:                        triangle1, triangle2 ← generate_triangles(vertices, "k+1")
26:                    end if
27:                end if
28:            end for
29:        end for
30:    end for
31: end Fonction
```

Ici, je ne détaille pas la génération des triangles ni la gestion de l'ouverture et de l'écriture dans le fichier. Dans le code, chaque triangle est généré "en dur" en fonction du voisin considéré, et les triangles ajoutés sont concaténés dans un grand vecteur de `string`, qui est ensuite écrit une seule fois dans le fichier. J'ai fait ceci car directement écrire dans le fichier pour chaque triangle était extrêmement long. **Étant donné que pour chaque voxel, les triangles sont générés en fonction des valeurs des voisins directement liés à ce voxel, et que chaque voisin ne sera vérifié qu'une seule fois (par exemple, le voisin à gauche de ce voxel ne sera jamais vérifié comme le voisin à droite d'un autre voxel), chaque triangle est créé une seule fois.**

3 Résultats sur des images 3D

(a) $S = 100$ (b) $S = 200$

FIGURE 1 – *engine.img* avec différentes valeurs de seuil S

(a) $S = 900$ (b) $S = 1100$

FIGURE 2 – *manix.img* avec différentes valeurs de seuil S

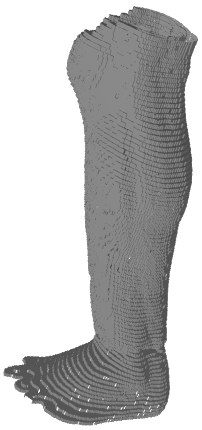

FIGURE 3 – *statueLeg.img* avec $S = 50$

4 Reconnaissance de l'image *whatisit*

À l'aide d'un seuil déterminé empiriquement, on observe que l'image *whatisit.img* est une dent.

(a) $S = 1000$ (b) $S = 800$

FIGURE 4 – *whatisit.img* avec différentes valeurs de seuil S

5 Code source

5.1 Image3D.hpp

```cpp
#include <iostream>
#include <vector>
#include <limits>
#include <algorithm>
#include <numeric>
#include <fstream>
#include <string>
#include <sstream>

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)];
    }
```

```
32
33      // Méthode pour définir la valeur d'un voxel
34      void set_value(int i, int j, int k, unsigned short value) {
35          data[get_index(i,j,k)] = value;
36      }
37  };
38
39  // Fonction pour charger une image
40  void load_image(char* image_read, Image3D &image3D) {
41      FILE* file = fopen(image_read, "r");
42      if (!file) {
43          std::cout<<"Erreur lors de l'ouverture de l'image"<<std::endl;
44          exit(EXIT_FAILURE);
45      }
46      // On lit de k en i pour respecter le balayage
47      for(int k = 0; k < image3D.dimZ; ++k) {
48          for(int j = 0; j < image3D.dimY; ++j) {
49              for(int i = 0; i < image3D.dimX; ++i) {
50                  unsigned char bytes[2]; // Ici, les voxels sont stockés sur
    2 octets
51                  if(fread(&bytes, sizeof(unsigned char), 2, file) != 2) { //
    Il faut indiquer à fread que l'on veut lire 2 octets
52                      std::cout<<"Erreur lors de la lecture de l'image"<<std::
    endl;
53                      exit(EXIT_FAILURE);
54                  }
55                  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)
56                  image3D.set_value(i,j,k,current_voxel_read);
57              }
58          }
59      }
60  }
61
62  struct Vertex {
63      float x,y,z;
64  };
65
66  // Fonction pour générer les sommets d'un voxel
67  std::vector<Vertex> generate_voxel_vertices(int i, int j, int k, float sizeX
    , float sizeY, float sizeZ) {
68      std::vector<Vertex> vertices(8);
69      vertices[0] = { (i - 0.5f) * sizeX, (j - 0.5f) * sizeY, (k - 0.5f) *
    sizeZ };
70      vertices[1] = { (i + 0.5f) * sizeX, (j - 0.5f) * sizeY, (k - 0.5f) *
    sizeZ };
71      vertices[2] = { (i + 0.5f) * sizeX, (j + 0.5f) * sizeY, (k - 0.5f) *
    sizeZ };
72      vertices[3] = { (i - 0.5f) * sizeX, (j + 0.5f) * sizeY, (k - 0.5f) *
    sizeZ };
73      vertices[4] = { (i - 0.5f) * sizeX, (j - 0.5f) * sizeY, (k + 0.5f) *
    sizeZ };
74      vertices[5] = { (i + 0.5f) * sizeX, (j - 0.5f) * sizeY, (k + 0.5f) *
    sizeZ };
75      vertices[6] = { (i + 0.5f) * sizeX, (j + 0.5f) * sizeY, (k + 0.5f) *
    sizeZ };
76      vertices[7] = { (i - 0.5f) * sizeX, (j + 0.5f) * sizeY, (k + 0.5f) *
    sizeZ };
```

```
77      return vertices;
78  }
79
80  void write_to_triangle(std::ostringstream &triangle, Vertex v1, Vertex v2,
        Vertex v3) {
81      triangle<<"facet normal 0 0 0"<<std::endl;
82      triangle<<"outer loop"<<std::endl;
83      triangle<<"vertex "<<v1.x<<" "<<v1.y<<" "<<v1.z<<std::endl;
84      triangle<<"vertex "<<v2.x<<" "<<v2.y<<" "<<v2.z<<std::endl;
85      triangle<<"vertex "<<v3.x<<" "<<v3.y<<" "<<v3.z<<std::endl;
86      triangle<<"endloop"<<std::endl;
87      triangle<<"endfacet"<<std::endl;
88  }
```

5.2 extraction_iso_surface.cpp

```
#include "Image3D.hpp"

for(int i = 1; i < dimX - 1; ++i) { // On évite les bords
    for(int j = 1; j < dimY - 1; ++j) {
        for(int k = 1; k < dimZ - 1; ++k) {
                unsigned short current_value = image3D.get_value(i,j,k);
                if(current_value > seuil) {
                    vertices = generate_voxel_vertices(i,j,k,sizeX,sizeY,
    sizeZ);
                    if (image3D.get_value(i-1,j,k) < seuil) {
                        std::ostringstream triangle1;
                        std::ostringstream triangle2;
                        write_to_triangle(triangle1,vertices[0],vertices[3],
    vertices[7]);
                        write_to_triangle(triangle2,vertices[7],vertices[4],
    vertices[0]);
                        triangles.push_back(triangle1.str());
                        triangles.push_back(triangle2.str());
                    }
                    if(image3D.get_value(i+1,j,k) < seuil) {
                        std::ostringstream triangle1;
                        std::ostringstream triangle2;
                        write_to_triangle(triangle1,vertices[1],vertices[2],
    vertices[6]);
                        write_to_triangle(triangle2,vertices[6],vertices[5],
    vertices[1]);
                        triangles.push_back(triangle1.str());
                        triangles.push_back(triangle2.str());
                    }
                    if(image3D.get_value(i,j-1,k) < seuil) {
                        std::ostringstream triangle1;
                        std::ostringstream triangle2;
                        write_to_triangle(triangle1,vertices[1],vertices[5],
    vertices[4]);
                        write_to_triangle(triangle2,vertices[4],vertices[0],
    vertices[1]);
                        triangles.push_back(triangle1.str());
                        triangles.push_back(triangle2.str());
                    }
                    if(image3D.get_value(i,j+1,k) < seuil) {
                        std::ostringstream triangle1;
                        std::ostringstream triangle2;
                        write_to_triangle(triangle1,vertices[3],vertices[7],
    vertices[6]);
                        write_to_triangle(triangle2,vertices[6],vertices[2],
    vertices[3]);
                        triangles.push_back(triangle1.str());
                        triangles.push_back(triangle2.str());
                    }
                    if(image3D.get_value(i,j,k-1) < seuil) {
                        std::ostringstream triangle1;
                        std::ostringstream triangle2;
                        write_to_triangle(triangle1,vertices[1],vertices[2],
    vertices[3]);
                        write_to_triangle(triangle2,vertices[3],vertices[0],
    vertices[1]);
                        triangles.push_back(triangle1.str());
                        triangles.push_back(triangle2.str());
```

```
                    }
                    if(image3D.get_value(i,j,k+1) < seuil) {
                        std::ostringstream triangle1;
                        std::ostringstream triangle2;
                        write_to_triangle(triangle1,vertices[5],vertices[6],
    vertices[7]);
                        write_to_triangle(triangle2,vertices[7],vertices[4],
    vertices[5]);
                        triangles.push_back(triangle1.str());
                        triangles.push_back(triangle2.str());
                    }
                }
            }
        }
    }

    for (const auto& tri : triangles) {
    stl_file<<tri;
    }

    stl_file<<"endsolid mesh"<<std::endl;
    stl_file.close();

    return 0;

}
```

N.B : Je ne sais pas pourquoi mais je ne peux pas mettre mon code entier dans mon document LaTeX, je reste à votre disposition pour vous donner tout le code.

6 Conclusion

Au terme de ce travail pratique, nous avons mis en œuvre une version simplifiée de l'algorithme du `Marching Cubes` pour extraire des iso-surfaces d'une image 3D, ce qui nous a permis de mieux comprendre les défis et les techniques associés à la visualisation de données médicales complexes. J'ai pris du plaisir car les résultats sont visuels.

Merci pour le temps et l'attention que vous avez consacrés à la lecture de ce compte-rendu.