Imagerie 3D

Compte rendu TP1

Stéphane Wouters

12 mars 2015

Table des matières

1 Stockage de l'image **3**
1.1 Lecture de l'image 3
1.2 Accès à une valeur x,y,z 3
1.3 Définition d'une valeur 4

2 Application **4**

3 Volume rendering **4**
3.1 Algorithme MinIP 4
3.2 Algorithme MIP 4
3.3 Algorithme AIP 5

4 Résultats **5**

5 What is it ? **6**

6 Code source **7**
6.1 Librairie Image 7
6.2 Fichier main 9

1 Stockage de l'image

Méthode : Utilisation d'une classe Image. Stockage dans un tableau binaire linaire, puis lecture via des calculs sur les indexes.

Tableau utilisé : *VALUE* bin ;*
Avec un *typedef unsigned short VALUE ;*

1.1 Lecture de l'image

L'image lue est entièrement stockée dans le tableau.

```
void load(char* filename) {
  FILE *f_image;

  if ((f_image = fopen(filename, "rb")) == NULL) {
    printf("\nErreur lecture %s \n", filename);
    exit(EXIT_FAILURE);
  }
  else {
    if (fread((VALUE*)bin, sizeof(VALUE), getSize(), f_image) !=
      (size_t)(getSize()))
    {
      printf("\nErrur lecture %s \n", filename);
      exit(EXIT_FAILURE);
    }
    fclose(f_image);
  }
}
```

1.2 Accès à une valeur x,y,z

L'accès à une valeur se fait par un calcul de dimension sur le tableau.

```
// Pour recuperer la valeur finale
VALUE getValue(int x, int y, int z) {
  return convertValue(*getVoxel(x,y,z));
}

// Convertion double octet
VALUE convertValue(VALUE v) {
  int x1 = v/256;
  int x2 = v - x1*256;
  return x2*256 + x1;
}

// Recuperer un poiteur vers le voxel
VALUE* getVoxel(int x, int y, int z) {
  return &bin[getIndexInTabBin(x,y,z)];
}

// Calcul d'exploration
int getIndexInTabBin(int x, int y, int z) {
  int v = z*sizeX*sizeY + y*sizeX + x;
  return v;
}
```

1.3 Définition d'une valeur

```
void setValue(int x, int y, int z, VALUE val) {
  *getVoxel(x,y,z) = convertValue(val);
}
```

2 Application

On test la classe en récupérant des valeurs de certains pixels, ainsi que les valeurs maximums et minimums.

```
Image in(448,576,72);
char path[250] = "beaufix.448x576x72.0.6250x0.6250x1.4.img";
in.load(path);
cout << "minValue = " << in.getMinValue() << endl;
cout << "maxValue = " << in.getMaxValue() << endl;
cout << "value = " << in.getValue(200,200,20) << endl;
```

On retrouve les valeurs spécifiés dans la feuille de TP.

— minValue = 0
— maxValue = 334
— l(200,200,20) = 14

3 Volume rendering

Pour calculer le volume rendering ; On crée une nouvelle image, puis on écrit sur la même couche numéro 0 le résultat du calcul appliqué sur toutes les couches en fonction de l'algorithme

3.1 Algorithme MinIP

```
for (int i = 0; i < in.sizeX; ++i)
  for (int j = 0; j < in.sizeY; ++j) {
    VALUE minVal = in.getValue(i,j,0);
    for (int k = 0; k < in.sizeZ; ++k) {
      minVal = min(in.getValue(i,j,k), minVal);
    }
    out.setValue(i,j,0,minVal);
  }
```

3.2 Algorithme MIP

```
for (int i = 0; i < in.sizeX; ++i)
  for (int j = 0; j < in.sizeY; ++j) {
    VALUE maxVal = in.getValue(i,j,0);
    for (int k = 0; k < in.sizeZ; ++k) {
      maxVal = max(in.getValue(i,j,k), maxVal);
    }
    out.setValue(i,j,0, maxVal);
  }
```

3.3 Algorithme AIP

```
for (int i = 0; i < in.sizeX; ++i)
  for (int j = 0; j < in.sizeY; ++j) {
    unsigned long long val = 0;
    for (int k = 0; k < in.sizeZ; ++k) {
      val += in.getValue(i,j,k);
    }
    out.setValue(i,j,0,val/in.sizeZ);
  }
```

4 Résultats

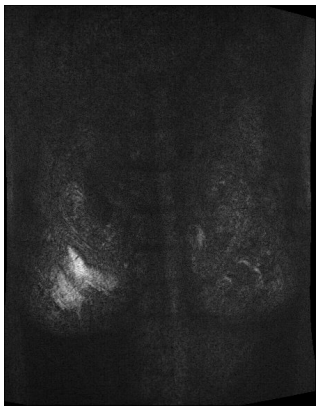

FIGURE 1 – Beaufix MinIP

FIGURE 2 – Beaufix MIP

FIGURE 3 – Beaufix AIP

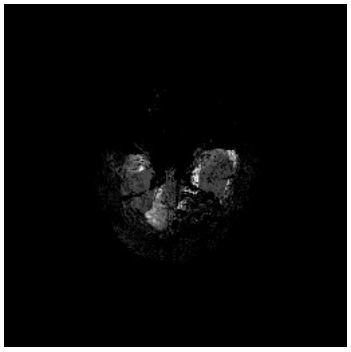

FIGURE 4 – Brainix MinIP

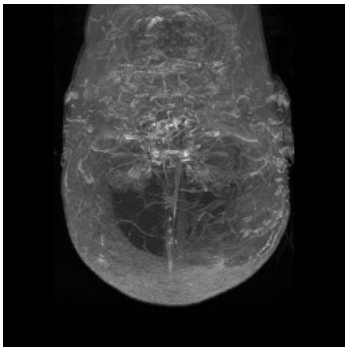

FIGURE 5 – Brainix MIP

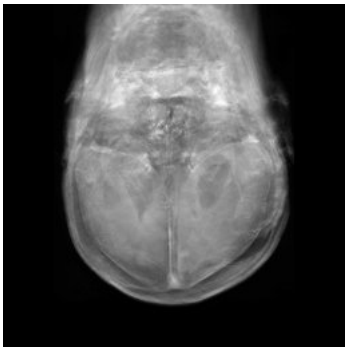

FIGURE 6 – Brainix AIP

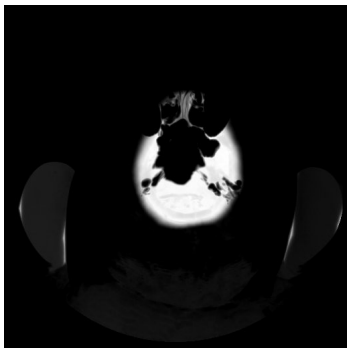

FIGURE 7 – Manix MinIP

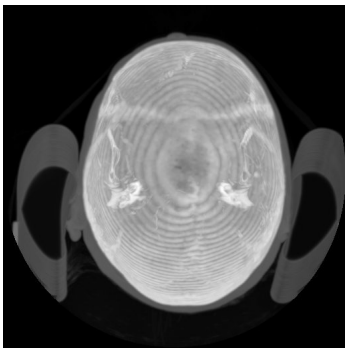

FIGURE 8 – Manix MIP

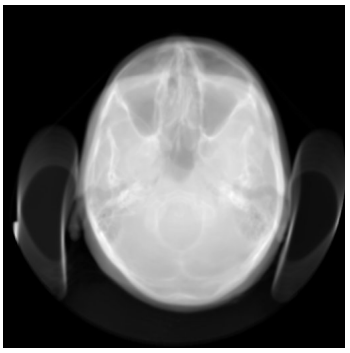

FIGURE 9 – Manix AIP

FIGURE 10 – Foot MIP

FIGURE 11 – Foot AIP

5 What is it ?

Avec l'algorithme AIP et en spécifiant au logiciel la taille de l'image, on obtient ceci :

FIGURE 12 – Whatisit AIP

6 Code source

6.1 Librairie Image

```
#ifndef IMAGE_PPM
#define IMAGE_PPM

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

typedef unsigned short VALUE;

class Image {

private:

  VALUE* bin;

  void allocateValues() {
    bin = new VALUE[getSize()];
  }

  int getSize() {
    return sizeX*sizeY*sizeZ;
  }

  void testValue(int x, int y, int z) {
    if (x > sizeX) {
      printf("L'indice se trouve en dehors des limites de l'image (x %d > %
          d)\n", x, sizeX);
      exit(0);
    }
    if (y > sizeY) {
      printf("L'indice se trouve en dehors des limites de l'image (y %d > %
          d)\n", y, sizeY);
      exit(0);
    }
    if (z > sizeZ) {
      printf("L'indice se trouve en dehors des limites de l'image (z %d > %
          d)\n", z, sizeZ);
      exit(0);
    }
  }

  VALUE convertValue(VALUE v) {
    int x1 = v/256;
    int x2 = v - x1*256;
    return x2*256 + x1;
  }

  VALUE unconvertValue(VALUE v) {
    return convertValue(v);
  }

  int getIndexInTabBin(int x, int y, int z) {
    int v = z*sizeX*sizeY + y*sizeX + x;
    return v;
```

```
54    }

56    VALUE* getVoxel(int x, int y, int z) {
        return &bin[getIndexInTabBin(x,y,z)];
58    }

60  public:

62    int sizeX; // Largeur
      int sizeY; // Hateur
64    int sizeZ; // Profondeur

66    Image(int sizeX, int sizeY, int sizeZ) {
        this->sizeX = sizeX;
68      this->sizeY = sizeY;
        this->sizeZ = sizeZ;
70      this->allocateValues();
      }
72
      VALUE getValue(int x, int y, int z) {
74      return convertValue(*getVoxel(x,y,z));
      }
76
      void setValue(int x, int y, int z, VALUE val) {
78      *getVoxel(x,y,z) = unconvertValue(val);
      }
80
      VALUE getMinValue() {
82      int min = bin[0];
        for (int i = 0; i < getSize(); ++i) {
84        int v = this->convertValue(bin[i]);
          if (v < min)
86          min = v;
        }
88      return min;
      }
90
      VALUE getMaxValue() {
92      int max = bin[0];
        for (int i = 0; i < getSize(); ++i) {
94        int v = this->convertValue(bin[i]);
          if (v > max)
96          max = v;
        }
98      return max;
      }
100
      void load(const char* filename) {
102     FILE *f_image;

104     if ((f_image = fopen(filename, "rb")) == NULL) {
          printf("\nPas d'acces en lecture sur l'image %s \n", filename);
106       exit(EXIT_FAILURE);
        }
108     else {
          if (fread((VALUE*)bin, sizeof(VALUE), getSize(), f_image) != (size_t)
              (getSize())) {
110         printf("\nErreur de lecture de l'image %s \n", filename);
            exit(EXIT_FAILURE);
112       }
          fclose(f_image);
```

```
    }
  }

  void write(const char* filename) {
    FILE *f_image;
    if ((f_image = fopen(filename, "wb")) == NULL) {
      printf("\nPas d'acces en ecriture sur l'image %s \n", filename);
      exit(EXIT_FAILURE);
    }
    else {
      if((fwrite((VALUE*)bin, sizeof(VALUE), getSize(), f_image)) != (
          size_t)(getSize())) {
        printf("\nErreur d'ecriture de l'image %s \n", filename);
        exit(EXIT_FAILURE);
      }
      fclose(f_image);
    }
  }
};

#endif
```

6.2 Fichier main

```
#include "Image.h"
#include <iostream>
#include <algorithm>

using namespace std;

void testRead() {
  Image in(448,576,72);
  char path[250] = "../ressources/BEAUFIX/beaufix.448x576x72.0.6250x0.6250
      x1.4.img";
  //char path[250] = "../ressources/BRAINIX/brainix.256x256x100.0.9375x0
      .9375x1.5.img";
  in.load(path);
  cout << "minValue = " << in.getMinValue() << endl;
  cout << "maxValue = " << in.getMaxValue() << endl;
  cout << "value = " << in.getValue(200,200,20) << endl;
}

void volumeRendering_MinIP(const char* path, int x, int y, int z) {
  Image in(x,y,z);
  in.load(path);

  Image out(x,y,1);

  for (int i = 0; i < in.sizeX; ++i)
  for (int j = 0; j < in.sizeY; ++j) {
    VALUE minVal = in.getValue(i,j,0);
    for (int k = 0; k < in.sizeZ; ++k) {
      minVal = min(in.getValue(i,j,k), minVal);
    }
    out.setValue(i,j,0,minVal);
  }

  char pathOut[256];
  sprintf(pathOut, "%s_MinIP.raw", path);
```

```
  out.write(pathOut);
}

void volumeRendering_MIP(const char* path, int x, int y, int z) {
  Image in(x,y,z);
  in.load(path);

  Image out(x,y,1);

  for (int i = 0; i < in.sizeX; ++i)
  for (int j = 0; j < in.sizeY; ++j) {
    VALUE maxVal = in.getValue(i,j,0);
    for (int k = 0; k < in.sizeZ; ++k) {
      maxVal = max(in.getValue(i,j,k), maxVal);
    }
    out.setValue(i,j,0, maxVal);
  }

  char pathOut[256];
  sprintf(pathOut, "%s_MIP.raw", path);
  out.write(pathOut);
}



void volumeRendering_AIP(const char* path, int x, int y, int z) {
  Image in(x,y,z);
  in.load(path);

  Image out(x,y,1);

  for (int i = 0; i < in.sizeX; ++i)
  for (int j = 0; j < in.sizeY; ++j) {
    unsigned long long val = 0;
    for (int k = 0; k < in.sizeZ; ++k) {
      val += in.getValue(i,j,k);
    }
    out.setValue(i,j,0,val/in.sizeZ);
  }

  char pathOut[256];
  sprintf(pathOut, "%s_AIP.raw", path);
  out.write(pathOut);
}

int main() {

  char engine[] = "../ressources/engine/engine.256x256x128.1x1x1.img";
  char foot[] = "../ressources/FOOT/foot.256x256x256.1.1.1.img";
  char beaufix[] = "../ressources/BEAUFIX/beaufix.448x576x72.0.6250x0.6250
      x1.4.img";
  char brainix[] = "../ressources/BRAINIX/brainix.256x256x100.0.9375x0.9375
      x1.5.img";
  char manix[] = "../ressources/MANIX/manixSansIV.512x512x48.0.4570x0.4570
      x3.0.img";
  char orange[] = "../ressources/ORANGE/orange.256x256x64.0.3906x0.3906x1
      .0.img";
  char whatisit[] = "../ressources/WHATISIT/whatisit.301x324x56.1.1.1.4.img
      ";

  volumeRendering_AIP(whatisit, 301, 324, 56);
```

```
  volumeRendering_MIP(whatisit, 301, 324, 56);
  volumeRendering_MinIP(whatisit, 301, 324, 56);
}
```