

Décompression du format VisualDEM

par [johann Sorel \(jsorel.developpez.com\)](http://jsorel.developpez.com)

Date de publication : 28/02/2007

Dernière mise à jour : 17/06/2007

Article orienté SIG.

Dans cet article vous verrez comment décompresser un fichier VisualDEM .dem. Ce format est ancien et peu utilisé, il stocke un modèle numérique de terrain.

Introduction

- 1 - Algorithme de décompression
 - 1.1 - Récupérer le géoréférencement
 - 1.2 - Passer l'entete
 - 1.3 - Récupérer chaque ligne compressée d'altitude
 - 1.4 - Decodage RLE
 - 1.5 - Ajustement des altitudes
 - 1.6 - On génère le tableau
- 2 - Classe complète Java
- 3 - Liens

Introduction

C'est quoi ce format que je n'ai jamais vue?

Vous entrez ici dans le monde des Systèmes d'Informations Géographiques, c'est un monde à part qui comprend plus de formats que de logiciels pour les traiter. **Pourquoi cette hypocrisie?** j'aimerais le savoir moi aussi, mais le fait est là, il en existe une quarantaine en format raster (en plus des formats que vous connaissez comme .bmp .png .jpg). Parmi ces formats certains sont des images satellites en infra-rouge, d'autres des cartes altimétriques, jusqu'à certain format temporel! Et oui on stocke l'évolution d'une image dans le temps, le format jpg semble bien pitoyable en comparaison. Certains stockent aussi des méta-données, on trouve parmi ces formats (MrSID, ECW) des taux de compression aussi évolués que le jpeg2000. Ces formats qui ne stockent que de simples images au premier abord sont pourtant pour les professionnels, et il faut les logiciels pour les ouvrir. On n'ouvre pas un fichier .ecw (très forte compression) d'un demi-gigaoctet avec paint ni avec adobe photoshop.

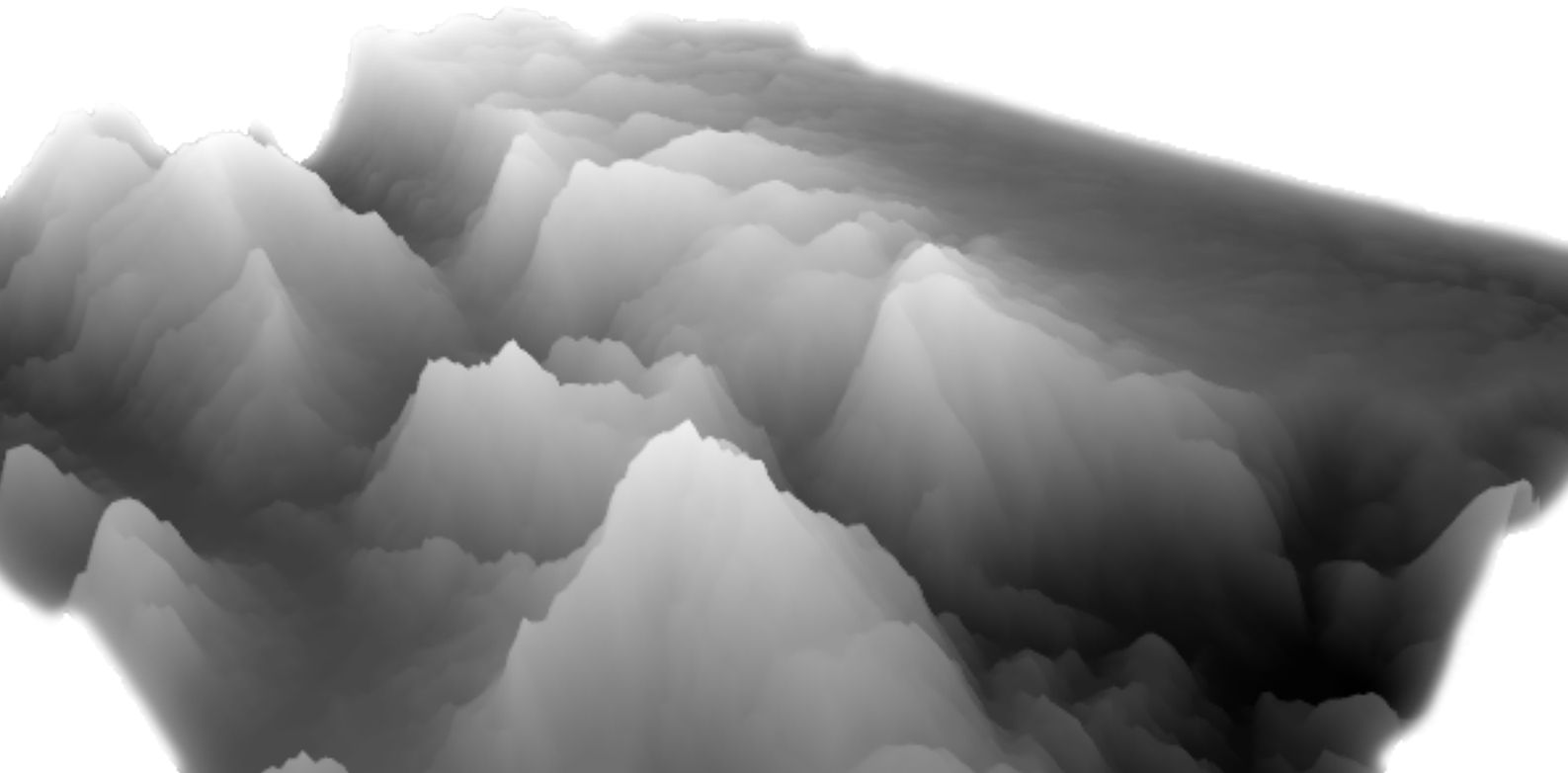
Maintenant que vous cernez mieux le domaine d'utilisation, voyons le format VisualDem .dem. Si vous avez eu peur je vous rassure, ce format ci date du temps des 486, il ne fera pas plus de 100Ko et aura une compression archaïque.

Commençons :

Tout d'abord le fichier a l'extension .dem ,il stocke un modèle numérique de terrain (MNT), on peut le qualifier de "carte d'altitude". La dalle stockée fait 258*258 points. (je parle de point car on ne peut pas parler ici de pixel) Le fichier stocke l'altitude en mètre, d'une valeur de 0 à 65535 mètres. Le pas entre chaque point d'altitude est de 75 mètres. Il y a une ligne ou colonne de recouvrement entre chaque dalle, ce qui nous fait une dalle réelle de 256*256 points et une couverture de 19.2 par 19.2 km. La taille du fichier varie entre 5Ko et 100Ko.

Les données sur le géoréférencement sont stockées au format texte dans le fichier .ctp

Pour la France, les dalles sont identifiables par leur nom qui commence par **FRAN** suivi de 2 lettres pour la colonne et 2 chiffres pour la ligne, la projection est en Lambert 2 étendu.



1 - Algorithme de décompression

1.1 - Récupérer le géoréférencement

Le Géoréférencement est expliqué dans un fichier .ctp, malheureusement, il est difficile de trouver ce fichier de nos jours.

On va donc faire un géoréférencement brut avec le nom du fichier pour la France. On connaît la taille de la dalle et comment elles sont placées (dallage sur toute la France). Le point d'origine des dalles est situé aux coordonnées $x=40000$ et $y=1690000$ sur une projection Lambert 2 étendu. Nous prendrons ce point comme origine. On sait qu'une dalle est un carré de 256 points d'altitude et qu'il y a 75 mètres entre 2 points.

Les premières valeurs 0 à récupérer sont la colonne et la ligne de notre dalle, une dalle "FRANAA01" a pour colonne 0 (AA) et ligne 0 (01).

L'algorithme (en java) :

```
adresse = "FRANB212.dem";
int pas = 75;
String coord = adresse.substring(adresse.length()-8,adresse.length()-4);
coordY = Integer.valueOf( coord.substring(coord.length()-2,coord.length()) ).intValue()
;

char c1 = coord.charAt(0);
c1 = Character.toUpperCase(c1);
char c2 = coord.charAt(1);
c2 = Character.toUpperCase(c2);

switch (c1) {
case 'A' : coordX = 0; break;
case 'B' : coordX = 26; break;
case 'C' : coordX = 52; break;
}

switch (c2) {
case 'A' : coordX += 0; break;
case 'B' : coordX += 1; break;
case 'C' : coordX += 2; break;
case 'D' : coordX += 3; break;
case 'E' : coordX += 4; break;
case 'F' : coordX += 5; break;
case 'G' : coordX += 6; break;
case 'H' : coordX += 7; break;
case 'I' : coordX += 8; break;
case 'J' : coordX += 9; break;
case 'K' : coordX += 10; break;
case 'L' : coordX += 11; break;
case 'M' : coordX += 12; break;
case 'N' : coordX += 13; break;
case 'O' : coordX += 14; break;
case 'P' : coordX += 15; break;
case 'Q' : coordX += 16; break;
case 'R' : coordX += 17; break;
case 'S' : coordX += 18; break;
case 'T' : coordX += 19; break;
}
```

```
case 'U' : coordX += 20; break;
case 'V' : coordX += 21; break;
case 'W' : coordX += 22; break;
case 'X' : coordX += 23; break;
case 'Y' : coordX += 24; break;
case 'Z' : coordX += 25; break;
}
```

Et il nous reste à faire l'ajustement :

```
coordX = coordX*pas*256 ;
coordY = (coordY-1)*pas*256 ;
System.out.println( coord + " X=" + coordX + " Y=" + coordY );
```

1.2 - Passer l'entete

Il n'y a pas d'information particulière dans l'entête, on peut donc la sauter sans souci.

Et on va stocker nos données dans une liste. On peut se le permettre car les fichiers dem ne sont pas volumineux, et ca simplifiera le décodage.

```
InputStream ips = new FileInputStream(adresse);

// on passe l'entete, 2048 octets
ips.skip(2048);

ArrayList<Integer> bloc_se = new ArrayList<Integer>();
// on stock notre bloc de données
while ( (val = ips.read()) != -1 ) {
    bloc_se.add(val);
}
ips.close();
```

1.3 - Récupérer chaque ligne compressée d'altitude

Maintenant que nous avons un bloc d'octet contenant uniquement nos données d'altitude, il va récupérer les lignes une à une. Une dalle contient 258 lignes (256 + 2 de recouvrements), pour récupérer la longueur de chaque ligne on prend les deux premiers octets. Ceux-ci indiquent le nombre N d'octets dédiés à la ligne.

On récupère les N octets dans une liste et on continue de la même manière jusqu'à avoir nos 258 lignes.

```
// on divise en ligne notre bloc
ArrayList<ArrayList<Integer>> bloc_pl = new ArrayList<ArrayList<Integer>>();
for( int X = 0 ; X < 258 ; X++ ) {
    longueur = bloc_se.get(n) * 256; n++;
    longueur = longueur + bloc_se.get(n); n++;

    ArrayList<Integer> rec = new ArrayList<Integer>();
    for( int i = 0 ; i < longueur ; i++ ) {
        rec.add(bloc_se.get(n));
    }
}
```

```
        n++;
    }

    bloc_pl.add(rec);
}
```

1.4 - Decodage RLE

Chacune des lignes que nous avons est compressée. Le codage est le suivant :

on prend le premier octet "VAL":

- il est supérieur ou égal à 128, on va recopier **257-VAL** fois l'octet suivant.

- il est inférieur à 128, on recopie les **VAL+1** octets suivant.

et on continue tant qu'on a des octets non decodés.

```
ArrayList<ArrayList<Integer>> bloc_dc = new ArrayList<ArrayList<Integer>>();

// on decode nos lignes
int total = 0;
for( int X = 0 ; X < 258 ; X++ ){

    ArrayList<Integer> arr = bloc_pl.get(X);
    ArrayList<Integer> rec = new ArrayList<Integer>();

    n = 0;

    do{

        val = arr.get(n);
        n++;
        total++;

        if( val >= 128 ){
            int decal = 257 - val;
            for( int t = 0 ; t < decal ; t++ ){
                rec.add(arr.get(n));
            }
            n++;
        } else{
            int decal = val + 1;
            for( int t = 0 ; t < decal ; t++ ){
                rec.add(arr.get(n));
                n++;
            }
        }
    }
    while ( n < arr.size() );

    bloc_dc.add(rec);
}
```

1.5 - Ajustement des altitudes

Maintenant nous avons les altitudes en relatif, c'est à dire que sur une ligne, chaque altitude est définie en fonction de la précédente. On va donner les vrais valeurs pour chaque point.

Les deux premiers octets indiquent l'altitude de départ.

Ensuite on prend chaque octet "VAL":

- il est supérieur à 128, **altitude = altitude - (256- val)**
- il est égal à 128, **altitude = (octet suivant)*256 + (octet 2*suivant)**
- il est inférieur à 128, altitude = altitude + val

Et continue jusqu'a 258 (ou qu'il n'y ai plus d'octet, le resultat est le meme).

```
// on ajuste l'altitude
for( int X = 0 ; X < 258 ; X++ ){
    ArrayList<Integer> arr = bloc_dc.get(X);
    ArrayList<Integer> rec = new ArrayList<Integer>();
    for( int i = 0 ; i < 258 ; i++ ){
        rec.add(0);
    }

    int raise = 0;
    n = 0;

    int altitude = arr.get(n) * 256 + arr.get(n + 1);
    n += 2;
    rec.set(raise, altitude);
    raise++;

    do{
        val = arr.get(n);
        n++;

        if( val == 128 ){
            altitude = arr.get(n) * 256 + arr.get(n + 1);
            n += 2;
        } else if( val < 128 )
            altitude += val;
        else if( val > 128 )
            altitude -= (256 - val);

        rec.set(raise, altitude);
        raise++;

        grid.test(altitude);
    }
    while ( n < arr.size() );

    bloc_alti.add(rec);
}
```

Voilà, nous avons maintenant une liste de liste 'bloc_alti' qui contient tous nos points d'altitude.

1.6 - On génère le tableau

On peut améliorer un peu cela, notamment pour un éventuel rendu 2D ou 3D.

```
float[][][] vertex = new float[258][258][3];
for( int X=0; X<258; X++ ){
    for( int Y=0; Y<258; Y++ ){
        vertex[X][Y][0] = (X*pas);
        vertex[X][Y][1] = (Y*pas);
        vertex[X][Y][2] = bloc_alti.get(Y).get(X);
    }
}
```

Il ne faut pas perdre de vue qu'on a travaillé avec la projection du format visualDem.

Pour ajuster le géoréférencement au lambert 2 étendu, ajouter 40000 en X et 1690000 en Y.

2 - Classe complète Java

```
public class DemMNT {

    private int longueur = 0;
    private int n = 0;
    private int val = 0;
    private int coordX = 0;
    private int coordY = 0;
    private ArrayList<ArrayList<Integer>> bloc_alti = new ArrayList<ArrayList<Integer>>();
    private float[][][] vertex = new float[258][258][4];
    private int pas = 75;

    public DemMNT( String adresse, GRIDMNT grid ){

        try{
            String coord = adresse.substring(adresse.length()-8,adresse.length()-4);
            coordY = Integer.valueOf( coord.substring(coord.length()-2,coord.length()) ).intValue();
        }

        char l1 = coord.charAt(0);
        l1 = Character.toUpperCase(l1);
        char l2 = coord.charAt(1);
        l2 = Character.toUpperCase(l2);

        switch (l1) {
            case 'A' :coordX = 0; break;
            case 'B' :coordX = 26; break;
            case 'C' :coordX = 52; break;
        }

        switch (l2) {
            case 'A' :coordX += 0; break;
            case 'B' :coordX += 1; break;
            case 'C' :coordX += 2; break;
            case 'D' :coordX += 3; break;
            case 'E' :coordX += 4; break;
            case 'F' :coordX += 5; break;
            case 'G' :coordX += 6; break;
            case 'H' :coordX += 7; break;
            case 'I' :coordX += 8; break;
            case 'J' :coordX += 9; break;
            case 'K' :coordX += 10; break;
            case 'L' :coordX += 11; break;
            case 'M' :coordX += 12; break;
            case 'N' :coordX += 13; break;
            case 'O' :coordX += 14; break;
            case 'P' :coordX += 15; break;
            case 'Q' :coordX += 16; break;
            case 'R' :coordX += 17; break;
            case 'S' :coordX += 18; break;
            case 'T' :coordX += 19; break;
            case 'U' :coordX += 20; break;
            case 'V' :coordX += 21; break;
            case 'W' :coordX += 22; break;
            case 'X' :coordX += 23; break;
            case 'Y' :coordX += 24; break;
            case 'Z' :coordX += 25; break;
        }
    }
}
```

```
coordX = coordX*pas*256 ;
coordY = (coordY-1)*pas*256 ;
System.out.println( coord + " X=" + coordX + " Y=" + coordY );

}catch(Exception e){
    JDialog dia = new JDialog();
    dia.setModal(true);

    Pgeoref p = new Pgeoref(dia,this);

    dia.getContentPane().add(p);

    dia.pack();
    dia.setResizable(false);
    dia.setLocationRelativeTo(null);

    dia.setVisible(true);
}

try{
    InputStream ips = new FileInputStream(adresse);

    //OK on passe l'entete, 2048 octets
    ips.skip(2048);
    System.out.println("taille entete : 2048");

    ArrayList<Integer> bloc_se = new ArrayList<Integer>();
    //OK on stock notre bloc de données
    while ( (val = ips.read()) != -1 ){
        bloc_se.add(val);
    }
    ips.close();
    System.out.println("taille bloc sans entete : " + bloc_se.size());

    //OK on divise en ligne notre bloc
    ArrayList<ArrayList<Integer>> bloc_pl = new ArrayList<ArrayList<Integer>>();
    long lng_diviser = 0;
    for( int X = 0 ; X < 258 ; X++ ){
        longueur = bloc_se.get(n) * 256;
        n++;
        longueur = longueur + bloc_se.get(n);
        n++;
        lng_diviser += longueur;

        ArrayList<Integer> rec = new ArrayList<Integer>();
        for( int i = 0 ; i < longueur ; i++){
            rec.add(bloc_se.get(n));
            n++;
        }

        bloc_pl.add(rec);
    }

    System.out.println("taille bloc dans lignes : " + lng_diviser);

    ArrayList<ArrayList<Integer>> bloc_dc = decodage(bloc_pl);
    ajustement(bloc_dc);

    updateCoords();

} catch( Exception e ){
    e.printStackTrace();
}
```

```
}  
  
private ArrayList<ArrayList<Integer>> decodage( ArrayList<ArrayList<Integer>> bloc_pl){  
    ArrayList<ArrayList<Integer>> bloc_dc = new ArrayList<ArrayList<Integer>>();  
  
    //OK on decode nos lignes  
    int total = 0;  
    for( int X = 0 ; X < 258 ; X++ ){  
  
        ArrayList<Integer> arr = bloc_pl.get(X);  
        ArrayList<Integer> rec = new ArrayList<Integer>();  
  
        n = 0;  
        total = 0;  
        do{  
  
            val = arr.get(n);  
            //System.out.print(val + ">");  
            n++;  
            total++;  
  
            if( val >= 128 ){  
                int decal = 257 - val;  
                total += decal;  
                //System.out.print(decal + " ");  
                for( int t = 0 ; t < decal ; t++ ){  
                    rec.add(arr.get(n));  
                }  
                n++;  
  
            } else{  
                int decal = val + 1;  
                total += decal;  
                //System.out.print(decal + " ");  
                for( int t = 0 ; t < decal ; t++ ){  
                    rec.add(arr.get(n));  
                    n++;  
                }  
            }  
        }  
        while ( n < arr.size() );  
  
        bloc_dc.add(rec);  
  
    }  
    //System.out.println("\n" + total);  
    return bloc_dc;  
}  
  
private void ajustement( ArrayList<ArrayList<Integer>> bloc_dc){  
  
    // on ajuste l'altitude  
    for( int X = 0 ; X < 258 ; X++ ){  
        ArrayList<Integer> arr = bloc_dc.get(X);  
        ArrayList<Integer> rec = new ArrayList<Integer>();  
        for( int i = 0 ; i < 258 ; i++ ){  
            rec.add(0);  
        }  
  
        int raise = 0;  
        n = 0;  
  
        int altitude = arr.get(n) * 256 + arr.get(n + 1);  
        n += 2;  
        rec.set(raise, altitude);  
        raise++;  
    }  
}
```

```
grid.test(altitude);

do{

    val = arr.get(n);
    n++;

    if( val == 128 ){
        altitude = arr.get(n) * 256 + arr.get(n + 1);
        n += 2;
    } else if( val < 128 )
        altitude += val;
    else if( val > 128 )
        altitude -= (256 - val);

    rec.set(raise, altitude);
    raise++;

    grid.test(altitude);

}
while ( n < arr.size() );

bloc_alti.add(rec);
}

private void updateCoords(){

    for( int X=0; X<258; X++ ){
        for( int Y=0; Y<258; Y++ ){
            vertex[X][Y][0] = (X*pas);
            vertex[X][Y][1] = (Y*pas);
            vertex[X][Y][2] = bloc_alti.get(Y).get(X);
        }
    }
}
```

3 - Liens

Il n'existe plus rien sur ce format, le seul lien que j'ai pu trouver est celui-ci.

Chez Éric Sibert

 **Un autre article pour le décodage du format VisualDEM**

