BUT-Info.3 2023-2024
Module "3 en 1"
Projet n°2

Format de compression d'image numérique

Interface publique du `CoDec`

Interface publique

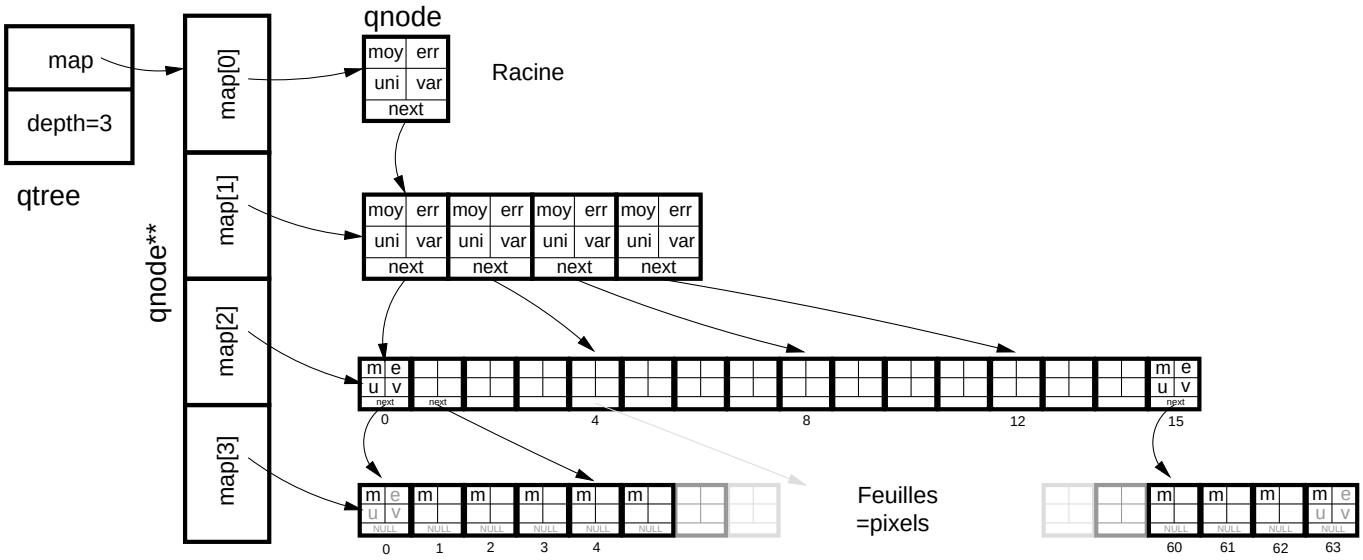
Vos `CoDec` doivent prendre la forme de bibliothèques (`qtc.h|libqtc.so`) inter-opérationnelles : vos programmes d'assemblage (partie exécutable, contenant la fonction `int main(int argc, char **argv)`) doivent pouvoir fonctionner (compilation, exécution) avec la lib. de quelqu'un d'autre et les fichiers au formats `qtc` encodés avec une version doivent être décodables par une autre.

Un certain nombre d'éléments doivent donc être standardisés, à commencer par les types et les prototypes des fonctions publiques constituant l'interface de la bibliothèque. La façon dont sont implémentées ces fonctions et la structuration des sources en modules est en revanche assez libre.

Les types publics

Il faut limiter leur nombre et leurs complexités. Pour ce projet nous utiliserons deux types publics : `qtree`, pour représenter la structure hiérarchique d'un `QuadTree` et `qnode`, pour un noeud de ce `QuadTree`.

Pour une image d'entrée (8x8), le `QuadTree` aurait ($4 = 1 + 2^3$) niveaux de tailles respectives 1|4|16|64 et sa représenation mémoire ressemblerait à la figure suivante.



types publics

```
typedef struct _node
{
  ushort moy;          /* pixel ou valeur différentielle [-255,+255] ramenée à [0,510]   */
  uchar  err;          /* erreur par rapport à la moyenne - sur 2 bits                    */
  uchar  uni;          /* bloc uniforme ou pas - sur 1 bit                                */
  double var;          /* variance                                                        */
  struct _node* next;  /* qnode '1° fils' dans le niveau suivant                          */
} qnode;

typedef struct
{
  qnode** map;         /* pointeur d'accès aux niveaux intermédiaires [0...depth]         */
  int depth;           /* hauteur de l'arbre - (depth+1) 'map' [0...depth]                */
} qtree;
```

Remarques

- pour une image source de tailles (2^nx2^n) le `QuadTree` à une `profondeur` (`depth`) de `n`, mais bien `(n + 1)` niveaux de la racine jusqu'aux "feuilles".
- les feuilles correspondent aux pixels (réorganisés) de l'image et seul leur champ `moy` est utilisé
- la réorganisation des pixels (cf. QTreeComp.2023.R1.pdf) permet de n'avoir qu'un pointeur par noeud : les 4 fils sont côte à côte.

Parcours

Cette structure permettra, comme toute structure d'arbre, deux types de *parcours récursifs en profondeur* :

ⓐ parcours **pré**fixe : on exécute une action sur le noeud courant puis on descend dans les noeuds fils
☞ ce type de parcours sera utilisé pour la *lecture* du `QuadTree` : filtrage (mode *lossy*), affichage, enregistrement (format `qtc` ou `pnm`), chargement depuis un fichier `qtc`...

ⓑ parcours **suf**fixe : on descend dans les noeuds fils puis, à la remontée, on exécute une action sur le noeud courant.
☞ ce type de parcours sera utilisé pour l'*écriture* du `QuadTree` : chargement depuis une image, passage en codage différentiel...

parcours préfixe

```
bool fprefix(qnode* node)
{
  if (!node) return false;
  /* 1°) d'abord on fait des choses sur <node>
   *     éventuellement on sort de la fonction */
  if (node->uni==1) return true;
  /* ... d'autres choses sur <node> ... */
  if (!node->next) return false;
  /* 2°) puis on descend récursivement */
  fprefix(node->next+0);
  fprefix(node->next+1);
  fprefix(node->next+2);
  fprefix(node->next+3);
  /* et c'est fini */
  return true
}
```

parcours suffixe

```
bool fsuffix(qnode* node)
{
  if (!node) return false;
  if (!node->next) return false;
  /* 1°) d'abord on descend récursivement */
  fsuffix(node->next+0);
  fsuffix(node->next+1);
  fsuffix(node->next+2);
  fsuffix(node->next+3);
  /* 2°) puis on fait des choses sur <node> */
  for (int i=0;i<4;i++)
    node->moy += (node->next+i)->moy;
  node->err = node->moy%4;
  node->moy = node->moy/4;
  return true
}
```

☞ une action utilisant un parcours dans sa version `encodeur` utilisera l'autre parcours dans sa version `décodeur`. Mais dans tous les cas le parcours commence à la racine (seul point d'entrée de l'arbre).

Les fonctions publiques

Elles aussi sont en nombre limité et ont des prototypes (paramètres / sortie) imposés.

les fonctions publiques

```
/* Gestion mémoire */
bool qtree_alloc(qtree **qt, int depth);              /* création d'un quadtree                */
bool qtree_free(qtree **qt);                          /* libération                            */
/* Conversion pixmap <> quadtree */
bool pixmap_to_qtree(G2Xpixmap *img, qtree **qt);     /* conversion pixmap  ->quadtree         */
bool qtree_to_pixmap(qtree* qt, G2Xpixmap **img);     /* conversion quadtree->pixmap           */
/* mode 'lossy' */
bool qtree_filter(qtree *qt, double α);               /* filtrage du quadtree                  */
/* Conversion quadtree standard <> différentiel */
bool qtree_diff(qtree *qt);                           /* passage en mode différentiel          */
bool qtree_undiff(qtree *qt);                         /* retour en mode standard               */
/* Ecriture sur fichier qtc */
bool qtree_fwrite_header(char *filename, int qtmode); /* écriture en-tête                      */
bool qtree_fwrite_Q1(qtree *qt, char *filename);      /* écriture quadtree filtré standard     */
bool qtree_fwrite_Q2(qtree *qt, char *filename);      /* écriture quadtree filtré différentiel */
/* Lecture d'une fichier qtc */
bool qtree_fread_header(char *filename, int *qtmode); /* lecture en-tête => magic_number       */
bool qtree_fread_Q1(qtree **qt, char *filename);      /* lecture fichier <.qtc> standard       */
bool qtree_fread_Q2(qtree **qt, char *filename);      /* lecture fichier <.qtc> différentiel   */
```

Structuration modulaire

Le découpage du projet en module reste assez libre, de même que le nombre, le status (`static|extern`) et les prototypes des fonctions non publiques (non accessibles à un utilisateur de la bibliothèque).

La seule contrainte est que l'utilisateur ait accès, via un *hearder* unique (`qtc.h`), à tous les types publics (y compris ceux issus d'une autre bibliothèque (`G2Xpixmap|bool|uchar...`), et à toutes les fonctions publiques.

Ressources annexes

Deux modules annexes (`utils.[h,c] | graphics.[h,c]`) vous sont fournis : ils contiennent quelques fonctions "hors programme", un peu plus complexes (cf. fichiers source `[h,c]` pour plus d'informations).

utils.h

```
#ifndef UTILS_H
  #define UTILS_H
  /*! pour les types (bool|uchar|...) et macros associées !*/
  #include <g2x_types.h>
  /*! renvoie le log. de base de l'entier n !*/
  uint log2_i(uint n);
  /*! Decoupe un chemin <path>=<dir/body.ext> en 3 morceaux <path:dir>|<body>|<ext> !*/
  void path_split(char* path, char** body, char** ext);
  /*! ECRITURE / LECTURE BIT A BIT DANS UN BUFFER BINAIRE */
  /*! utilitaire : écrit sur la sortie standard d'erreur <stderr> */
  void printnbin(uchar x, int n, char *sep);
  /*! Ecriure d'un bloc de <nbit> à une adresse donnée (buffer) !*/
  void push_bits(uchar **dest, uchar* src, int nbit, size_t *wbit);
  /*! Lecture d'un bloc de <nbit> dans une adresse donnée (buffer) !*/
  void pull_bits(uchar **dest, uchar* src, int nbit, size_t *rbit);
  /*! version 'simul' de <pushbits> se contente d'évaluer les tailles des données à écrire !*/
  void countbits(size_t nbit, size_t* wbyte, size_t *wbit);
#endif
```

qtgraphics.h

```
#ifndef _QTGRAPHICS_H
  #define _QTGRAPHICS_H
  #include <qtc.h>
  void qtree_show_grid(qtree* qt);              /*! grille de segmentation   !*/
  void qtree_show_bloc(qtree* qt);              /*! image quadtree           !*/
  void qtree_show_histo(int *Hist, int nsymb);  /*! !histogramme             !*/
#endif
```

☞ Attention :

le module `utils.[h,c]` fera nécessairement partie de la bibliothèque `[qtc.h|libqt.so]` puisqu'il contient les fonctions de lecture/écriture bit à bit indispensable pour la gestion des fichiers `.qtc`.

le module `graphics.[h,c]` n'en fait pas partie : c'est l'utilisateur qui décide ce qu'il veut visualiser et sous quelle forme. Ce module vous est fourni car il y a quelques subtilités liées à l'interfaçage `libg2x|OpengGl`.