Daouda Kanoute

Rapport n°27 : Compression par fenetres

Algorithmique Avancée

---

1. Énoncé du projet

Pour le cours d'Algorithmique avancée, il m'a été de mandé de faire un programme de compression par perte, voici l'énoncé :

Le principe est, ici, brutal : il s'agit de parcourir l'image par fenêtres de tailles nxn et conservant uniquement un groupe de quatre pixels pour chaque fenêtre. Choisir, pour chaque fenêtre, efficacement, les couleurs de ces quatre pixels. Dans ce cas le ratio est connu, c'est 4/(nxn). Par contre ce qui ne l'est pas, c'est la qualité de l'image produite. Mettre en place un système de mesure de la qualité de l'image (on peut chercher sur le web). Mettre en place une procédure subjective, questionnaire, de mesure de la qualité de l'image produite. Discuter les résultats produits.

---

2. Compression par pertes

La compression avec perte est l'acte de supprimé certaines donnée, pour alléger l'image, ces donnée sont considéré comme non nécessaire car l'impact de leurs perte ne changera par grandement la perception que l'on à de l'image (.jpg), cependant l'énoncé de ce projet est quelque peu plus radicale selon la taille que l'on donne à la fenetre les pertes apporté peuvent facilement changé totalement la perception que l'on peut avoir de l'image.

pour vous aider à y voir plus claire voici quelque exemple :

 sans modification

 image taille 4*4

 image taille 6*6

Parcours de l'image par fenetre de taille n*n

La premiere chose à faire est de parcours l'image étant une grille de pixels, pour cela j'ai mis deux boucle j et k. Chaque fois que le nombre de pixels voulu sur une ligne à été touché, on passera à la suivante jusqu'a remplir la fenetre n.

```
void compression(Image *i, int n)
{
  // Parcours des blocs de n*n pixels de l'image
  for (int j = 0; j < i->sizeY - n + 1; j += n)
  {
    for (int k = 0; k < i->sizeX - n + 1; k += n)
    {
      // Calcul de la valeur moyenne du bloc
```

```
      int v = couleur_efficace(i, j, k, n);
      GLubyte val = (GLubyte)v;
      // Mise à jour des pixels du bloc avec la valeur moyenne
      update_bloc(i, j, k, n, val);
    }
  }
}
```

Garder 4 pixels

Pour cela j'ai implémenter la fonction update_bloc qui parcours la fenetre n*n qu'elle prendra de la fonction compression(), puis conserva 4 pixels de cet fenetre et mettra les pixels inutilisé à 0, cela n'est pas vraiment une compression car au final ces pixels meme si transparente existeront toujours.

```
{
  // Parcours des pixels du bloc
  for (int y = 0; y < n; y++)
  {
    for (int x = 0; x < n; x++)
    {
      if ((y == 0 || y == n - 1) && (x == 0 || x == n - 1))
      {
        // Garde seulement les 4 coins de la fenêtre
        int x2 = k * n + x;
        int y2 = j * n + y;
        if (x2 >= 0 && x2 < i->sizeX && y2 >= 0 && y2 < i->sizeY)
        {
          GLubyte *imr = i->data + 3 * (y2 * i->sizeX + x2);
          GLubyte *img = imr + 1;
          GLubyte *imb = imr + 2;
          *imr = val;
          *img = val;
          *imb = val;
        }
      }
      else
      {
        // Mise à jour des autres pixels avec la valeur 0
        GLubyte *imr = i->data + 3 * (j * i->sizeX + k + y * i->sizeX + x);
        GLubyte *img = imr + 1;
        GLubyte *imb = imr + 2;
        *imr = 0;
        *img = 0;
        *imb = 0;
      }
    }
  }
}
```

Les 4 pixels prendront la valeurs Val passé en parametres dans cet example ils prendront la valeur "val" qui est égal à couleur_efficace( ).

Choisir efficacement les couleurs

J'estime que cela signifie qu'il faut choisir les couleurs qui feront perdre le moins d'information importante à l'image et cela peut passer par différentes approches, je me suis concentré sur trois d'entre elles.

Couleurs moyennes : En choisissant la moyenne des quatre pixels choisies sur chaque fenêtre, ou en faisant la moyenne des couleurs aux alentours de ces pixels.

Couleurs dominante : En déterminant la couleur la plus utilisée dans la fenêtre et en l'appliquant aux quatre pixels, on "s'assure" de garder le gros de l'information.

Couleurs extrêmes : en faisant la moyenne des couleurs pour appliquer la valeur la plus éloignée de cette moyenne.

Toutes ces méthodes ont le même gros problème qui est que les informations deviennent de moins en moins exacte, selon la taille de la fenêtre choisie, j'ai décidé d'implémenter une fonction qui renvoie la couleurs dominante car je trouve cela plus beau.

---

mesure de la qualité de l'image

il existe plusieurs façons de mesurer la qualité d'une image produite après une réduction de résolution. L'une des méthodes les plus utilisées est l'indice de qualité de l'image Qualité perceptuelle d'images, qui mesure la similarité visuelle entre l'image originale et l'image réduite. Un autre indice couramment utilisé est le taux de distorsion structurée (SDR), qui mesure la différence entre l'image originale et l'image réduite en tenant compte de la structure de l'image, il en existe plusieurs plus ou moins poussé.

Ma fonction de mesure de qualité, prend en argument 2 images et les comparent puis renvoie le nombre de pixel différend.

mesure de la qualité de maniere subjectif

Pour mesurer la qualité de l'image de maniere subjectif un petit questionnaire de 6 question à été fait, pour me permettre d'avoir une idée global de la qualité de compression.

- La qualité visuelle de l'image est-elle acceptable ? L'image est-elle floue, ou a-t-elle des défault visibles ?
- La résolution de l'image a-t-elle été affectée par la compression ? L'image est-elle plus petite ou plus granuleuse qu'avant la compression ?
- La luminosité et la balance des couleurs de l'image ont-elles été affectées par la compression ? L'image est-elle trop sombre ou trop claire, ou les couleurs sont-elles déformées par rapport à l'original ?
- La gamme de couleurs de l'image a-t-elle été affectée par la compression ? Y a-t-il des couleurs manquantes ou inattendues dans l'image compressée par rapport à l'original ?
- La précision des détails de l'image a-t-elle été affectée par la compression ? Les détails fins tels que les lignes et les bords sont-ils flous ou estompés dans l'image compressée par rapport à l'original ?
- De maniere globale que pensez-vous de l'image le resultat est-il agréable à l'oeil ?

Selon la majorité des participants, est trop altéré, meme si on peut y comprendre les informatition globale cela reste tout de meme assez problématique.

Tout les participants ont remarqué une diminution de la résolution de l'image suite à la compression.

Beaucoup de participant trouvent que la luminosité à grandement baissé tandis qu'une minorité n'y voit pas de difference a ce niveau là.

Lors de la décompression la couleurs rouge est trop exploité. De maniere globale il à été dit que les pixels vides posaient un probleme, meme si cela n'alterait pas énormément leurs perception de l'image.

Probleme rencontrée

```
void update_bloc(Image *i, int j, int k, int n, GLubyte val)
{
  // Parcours des pixels du bloc
  for (int y = 0; y < n; y++)
  {
    for (int x = 0; x < n; x++)
    {
      if ((y == 0 || y == n - 1) && (x == 0 || x == n - 1))
      {
        // Garde seulement les 4 coins de la fenêtre
        int x2 = k * n + x;
        int y2 = j * n + y;
        if (x2 >= 0 && x2 < i->sizeX && y2 >= 0 && y2 < i->sizeY)
        {
          GLubyte *imr = i->data + 3 * (y2 * i->sizeX + x2);
          GLubyte *img = imr + 1;
          GLubyte *imb = imr + 2;
          *imr = val;
          *img = val;
          *imb = val;
        }
      }
      else
      {
        // Mise à jour des autres pixels avec la valeur 0
        GLubyte *imr = i->data + 3 * (j * i->sizeX + k + y * i->sizeX + x);
        GLubyte *img = imr + 1;
        GLubyte *imb = imr + 2;
        *imr = 0;
        *img = 0;
        *imb = 0;
      }
    }
  }
}
```

Version 2 :update_bloc

```
void update_bloc(Image *i, int j, int k, int n, GLubyte val)
{
```

```
  // Parcours des pixels du bloc
  for (int y = 0; y < n; y++)
  {
    for (int x = 0; x < n; x++)
    {
      if (y % 2 == 0 && x % 2 == 0)
      {
        // Garde seulement toutes les lignes et colonnes paires de pixels
        GLubyte *imr = i->data + 3 * (j * i->sizeX + k + y * i->sizeX + x);
        GLubyte *img = imr + 1;
        GLubyte *imb = imr + 2;
        *imr = val;
        *img = val;
        *imb = val;
      }
      else
      {
        // Mise à jour des autres pixels avec la valeur 0
        GLubyte *imr = i->data + 3 * (j * i->sizeX + k + y * i->sizeX + x);
        GLubyte *img = imr + 1;
        GLubyte *imb = imr + 2;
        *imr = 0;
        *img = 0;
        *imb = 0;
      }
    }
  }
}
```

 Version 2 :update_bloc

La fonction update_bloc() est la fonction principale de mon programme, son rôle est de mon programme son role est de parcourir chaque bloc qu'il reçoit et de garder 4 pixels dans ce bloc et de supprimer le reste. Mais je n'ai pas reussi à le faire correctement, c'est la raison pour laquelle les pixels inutiles sont mis à 0. J'ai eu plusieurs idée pour regler ce probleme.

La premiere étant de parcourir toutes l'images apres le passage de la fonction pour décaller tout les pixels non-nulle vers la gauche, mais plusieurs problème se posent alors, c'est très long, car il faut parcourir tout les pixels pour pouvoir ce déplacement, et cela ne règle pas le problème et ne fait que le cacher.

Pour ces raisons j'ai opté pour une autre méthode qui consiste à creer un nouveaux tableaux de pixels, qui ne prendrait que les pixels utilisé et mettra à jour l'ancien tableaux.

```
    void enlever_pixels_0(Image *i)
{
  GLubyte *pixels[i->sizeX * i->sizeY];
  int nb_pixels = 0;

  for (int y = 0; y < i->sizeY; y++)
  {
    for (int x = 0; x < i->sizeX; x++)
```

```
    {
      GLubyte *pixel = i->data + 3 * (y * i->sizeX + x);
      if (*pixel != 0)
      {
        pixels[nb_pixels++] = pixel;
      }
    }
  }

  GLubyte *new_data = malloc(sizeof(GLubyte) * nb_pixels * 3);
  memcpy(new_data, pixels, sizeof(GLubyte) * nb_pixels * 3);
  free(i->data);

  i->data = new_data;
  int racine = sqrt(nb_pixels);
  i->sizeX = racine;
  i->sizeY = racine;
}
```

Cela à en quelque sorte fonctionné mais le résultat n'était pas ce que j'attendait l'image était beaucoup trop altérer.

 enlever_pixels_0

Le fait que la fonction update_bloc() ne fonctionne pas correctement pose aussi probleme pour les autres fonctions car cela signie qu'il n'y a que 4 pixels par fenetres qui prendront les valeurs demander tandis que le reste sera noir.