Rapport Projet Problème scientifique informatique

Matthéo Pho, Urbain Lantrès, Daphné Perrard

TD N

Lors de la réalisation de ce projet nous avons dû concevoir plusieurs classes différentes :

- Pixel

  La classe Pixel est composé de trois attributs byte r, g et b correspondant respectivement au composante rouge, vert et bleue d'un pixel.

  Nous avons réalisé trois constructeurs liés à cette classe un avec en entrée trois composantes de couleurs permettant de créer un pixel avec ses trois composantes, un sans entrée créant un pixel noir, et le troisième prenant en entrée une valeur et créant un pixel avec pour composantes r, g et b cette valeur.

  Notre classe Pixel est composé des méthodes :

  Pixel Greyscale () : permettant de calculer la valeur en niveau de gris d'un pixel

  Pixel Copy () : effectuant la copie d'un pixel

  override string ToString() : effectuant la réalisation textuelle d'un pixel

  bool Equals(object p) : testant si deux pixels sont égaux (si leurs composantes r, g et b sont identiques

- MyImage :

  La classe MyImage est composé de différents attributs informant sur la taille de l'image, la taille du fichier, la position du premier pixel du fichier, la taille de la deuxième partie du header la largeur et la hauteur de l'image, le nombre d bits par pixel et la création du tableau du header ainsi que celui des pixels ; ainsi qu un attribut FlipMode de type enum.

  Nous avons réalisé deux constructeurs, un créant une instance à partir d'un fichier référencié par le chemin d'accès, l'autre créant une instance à partir d'une image récupérée depuis un flux.

  Notre classe MyImage est composée de 4 constructeurs créant une instance à partir d'un fichier référencé par le chemin d'accès, à partir d'un flux, à partir d'une largeur et hauteur entrée en paramètre, en copiant une instance existante ;

  Notre classe MyImage est composé des méthodes suivantes :

  private void ConsumeStream(Stream stream) : utilise le flux pour remplit les attributs d'une image

  private int _position(int x, int y) : Calculant la position du 1er octet représentant le pixel à une position en entrée

  public Pixel this[int x, int y]: récupérant le pixel a une position en entrée

  public void Save(string filename) : Sauvegarde une image dans le fichier indiqué par le nom en entrée

  public MyImage Greyscale() : transformant une image en nuance de gris

  public MyImage BlackAndWhite() : transformant une image en noir et blanc

  public MyImage Negative() : retournant le négatif d'une image

  public MyImage Invert() : retournant une copie inversé d'une image

  public MyImage HideImageInside(MyImage imageToHide) : Cachant une image dans une autre

  public MyImage Histogram(bool channel_r = true, bool channel_g = true, bool channel_b = true): produisant l'histogramme d'une image

  public MyImage GetHiddenImage() : récupérant l'image cache dans l'image

  public MyImage Rotate(int angle): tournant l'image de l'angle en entrée

  public MyImage Scale(float echelle) : rétrécissant l'image selon le facteur en entrée

  public override bool Equals(object i): vérifiant l'égalité entre deux images.

- StreamExtended :

  Notre classe StreamExtended est composée des méthodes suivantes :

  static byte[] ReadBytes(this Stream stream, int length): permettant de lire en nombre 'length' d'octets depuis le flux d'entrée

  static byte[] ExtractBytes(this byte[] array, int length, int offset = 0): permettant d'extraire une partie du tableau d'octets

  static void InsertBytes(this byte[] array, byte[] data, int offsetTo = 0, int offsetFrom = 0, int length = -1): permettant de modifier un tableau d'octets avec un autre tableau d'octets

- Convolution

  Pour la réalisation de notre classe convolution nous avons créé un dictionnaire nous permettant d'utiliser plus facilement le dictionnaire kernel, nous avons mis dans ce dictionnaire les matrices de détection des bords, de flou, de flou de Gauss en 3x3 et en 5x5. Nous avons aussi 3 attributs de type enum EdgeProcessing, KernelOrigin et Kernel.

  Notre classe Convolution est composée des méthodes suivantes :

  static MyImage ApplyKernel(this MyImage image, float[,] kernel, KernelOrigin origin = KernelOrigin.Center, EdgeProcessing KernelOrigin. Center = EdgeProcessing.KernelCrop) : appliquant une convolution à une image en entrée

  Nous avons appliqué à la méthode ApplyKernel une surcharge permettant d'effectuer la fonction de manière plus rapide :

  static MyImage ApplyKernel(this MyImage image, Kernel kernel, KernelOrigin origin = KernelOrigin.Center, EdgeProcessing edgeProcessing = EdgeProcessing.KernelCrop):

- Utils

  La classe Utils est composé de plusieurs méthodes :

  static uint LittleEndianToUInt(byte[] input, int offset = 0) : transformant un tableau d'octets en une valeur de type uint

  static int LittleEndianToInt(byte[] input, int offset = 0):transformant un tableau d'octets en une valeur de type int

  static ushort LittleEndianToUShort(byte[] input, int offset = 0): transformant un tableau d'octets en une valeur de type ushort

  static byte[] IntToLittleEndian(int input) : transformant une valeur de type int en un tableau d'octets

  static byte[] UIntToLittleEndian(uint input):transformant une valeur de type uint en un tableau d'octets

  static byte[] UShortToLittleEndian(ushort input) : transformant une Valeur de type ushort en un tableau d'octets

- FractalGen

  Notre classe FractalGen ne contient qu'une seule méthode **static MyImage GenerateFractal(int width, int height, Complex c)** permettant de générer une fractale de Julia à partir du nombre complexe c.