Master 2 Informatique | CIM

Programmation créative - 2018-2019

TD 1

Processing TD1

Premiers pas

Le but de cet exercice est de vous familiariser avec Processing.

Quelques raccourcis intéressants :

Ctrl + R : Exécuter le programme
Ctrl + Maj + T : "Tweak" : permet de modifier la valeur des variables et voir le résultat en direct
Ctrl + Espace : complétion intelligente
Ctrl + T : indentation automatique du code

Documentation : `https://processing.org/reference`

Premier programme

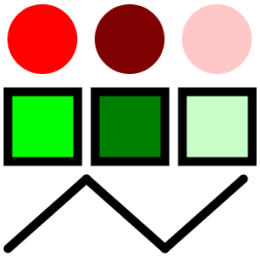
```
void setup() {
  //Fonction appelee a l'initialisation
}

void draw() {
  //Fonction appelee en boucle
}
```

Questions :

1. Dans la fonction `draw()`, dessinez un cercle centré en (50,50) et de rayon 10.
2. Remplissez-le avec du rouge.
3. Exécutez votre code (Sketch > Exécuter).
4. Exécutez votre code en mode **Tweak** (Sketch > Tweak).
   Des curseurs apparaissent au niveau des valeurs des variables rayon et couleur. Déplacez votre souris sur une valeur, puis augmentez-la par un mouvement vers la droite, ou diminuez-la par un mouvement vers la gauche.

1 Synthèse d'image

Exercice 1 : Primitives géométriques

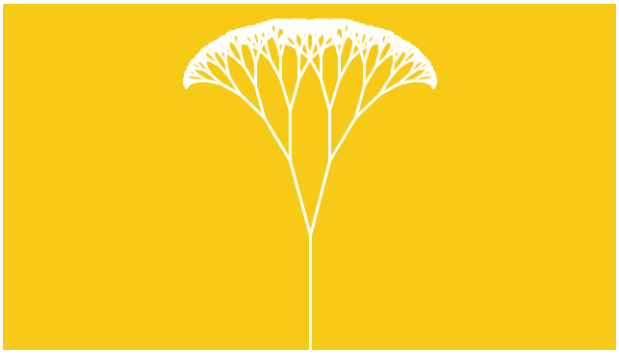


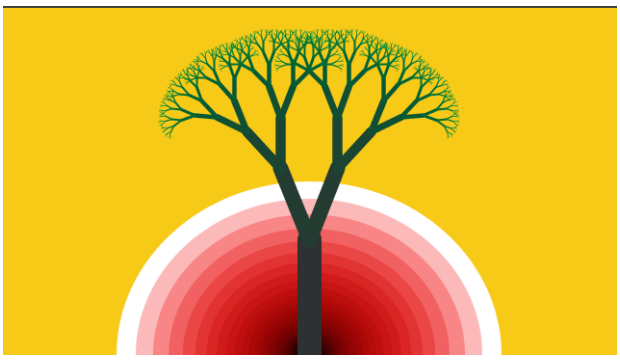
Le but de cet exercice est de vous faire manipuler les primitives géométriques (cercle, rectangle, ligne) afin de réaliser un dessin simple.

Dans le fichier fourni **ellipses.pde**, modifiez le code afin d'obtenir un résultat similaire à celui présenté dans la figure de droite :

1. Changez le mode de dessin du rectangle de manière à ce qu'il soit basé sur son centre : `rectMode(CENTER)`.
2. Sous les trois cercles, ajoutez trois rectangles de taille identique.
3. Colorez ces rectangles de façon à réaliser un dégradé de vert : du plus intense à gauche, au plus pâle à droite.
4. Ajoutez une bordure à ces rectangles.
5. Fixez l'épaisseur de la bordure à 3 pixels : `strokeWeight(float w)`
6. Sous les trois rectangles, dessinez un zig-zag en utilisant trois lignes différentes.

Exercice 2 : Dessin récursif





Ouvrez le fichier **trees.pde**.

Exécutez-le : un arbre similaire à celui affiché dans la figure de gauche ci-dessus apparaît. Bouger la souris de droite à gauche : l'angle d'inclinaison des branches varie.

Il s'agit d'un arbre dessiné récursivement : chaque ligne se divise en deux lignes de longueur moindre.

Lisez attentivement le code, puis modifiez-le de façon à obtenir un résultat similaire à celui présenté dans la figure de droite :

1. Changez la couleur de l'arbre (**ex :** marron).
2. Changez l'épaisseur du trait afin qu'il soit plus épais à la base de l'arbre, et plus fin à son sommet. Vous pouvez choisir l'épaisseur du trait comme 0.2 fois la longueur de la branche `h`.
3. Effectuez un dégradé de couleur sur l'arbre : de marron à sa base, à vert à son sommet. Pour cela, modifiez la fonction `tree()` afin qu'elle accepte le paramètre de couleur : `tree(float h, color c)`. **Indice :** Plus on se rapproche du sommet de l'arbre, plus la valeur du canal vert augmente, et plus les valeurs des canaux rouge et bleu diminuent.
4. Ecrivez une fonction récursive `soleil(float rayon)` qui permette de dessiner un soleil en **arrière-plan**, où chaque cercle contient un cercle plus petit (cf. figure de droite).
5. Dans la fonction `soleil()`, faites un dégradé (**ex :** de rouge) du cercle le plus petit au plus grand. Modifiez pour cela la signature de la fonction : `soleil(float rayon, color c)`.

2 Analyse d'image

Prise en main :

L'objet `PImage` permet de manipuler des images dans Processing.

Ouvrez et testez le programme **blank_image.pde**. Une image vide est créée dans la fonction `setup()`. Une fenêtre avec un fond blanc s'affiche.

Question : Affichez l'image vide dans cette fenêtre.

Exercice 1 : Calcul d'histogramme

Ouvrez le fichier **histogram_grey.pde**. A l'exécution, une fenêtre avec une image de feuille doit s'ouvrir.

(a) Affichage initial de l'image

(b) Histogramme superposé à l'image

Questions :

1. Calculez l'histogramme dans la fonction `histo()`. On rappelle le pseudo-algorithme de calcul de l'histogramme :

   (a) Initialisation d'un tableau `hist` pouvant contenir 256 valeurs

   (b) Pour chaque pixel de l'image :

      i. Récupérer son intensité `I`

      ii. Incrémenter `hist[I]` de 1

   Pour récupérer l'intensité d'un pixel, utilisez les fonctions `img.get(x,y)` et `brightness(color)` (consultez la documentation au besoin).
   NB : Le tableau `hist` à compléter est déjà déclaré dans le code, et la fonction `drawHisto()` déjà fournie permet d'afficher l'histogramme par-dessus l'image (cf. figure de droite).

2. Dans une nouvelle fonction `negatif()`, calculez le négatif de l'image.
   NB : N'oubliez pas les appels à `loadPixels()` et `updatePixels()` !

3. Affichez l'histogramme du négatif de l'image. Que constatez-vous ?

Exercice 2 : Égalisation d'histogramme

Ouvrez le fichier **histogram_equalization.pde**. A l'exécution, une fenêtre avec une image sombre doit s'ouvrir. L'objectif de cet exercice est de corriger le contraste de cette image par égalisation d'histogramme.

Lisez attentivement le code fourni.

Questions :

1. Dans la fonction `egalisation()`, calculez l'image égalisée (`imgEgalisee`). On rappelle le pseudo-algorithme d'égalisation :

   (a) Calcul de l'histogramme cumulé $\texttt{hist}_{\texttt{cumul}}$

   (b) Pour chaque pixel d'intensité `I`, associer une nouvelle intensité $\texttt{I}' = 255 * \texttt{hist}_{\texttt{cumul}}[\texttt{I}]$

NB : L'histogramme cumulé est déjà calculé.

2. L'image égalisée est toutefois très bruitée par la présence de pixels blancs aberrants. Nous souhaitons seuiller l'image afin de supprimer le bruit.

   Dans la fonction `seuillage()`, calculez l'image seuillée (`imgSeuillee`) à partir de l'image égalisée. Mettre les pixels au-dessus du seuil à la couleur noire, sinon conserver leur intensité originale.

   Ce seuil permet-il de supprimer tout le bruit et de conserver les détails ?