IV – Expériences Virtuelles par WebCam et Kinect

46

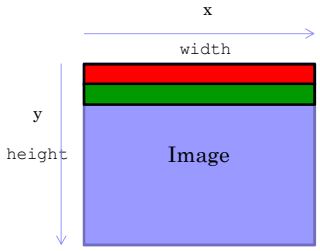
IV.1 – Principe de fonctionnement d'une caméra webCam

- Caractéristiques :
  - Nom de la caméra : « Logitech HD Webcam C310 »
  - Taille des images :
    - plusieurs tailles possibles
    - tailles connues en interrogeant la caméra
    - 640x480, ...
  - Vitesse d'acquisition :
    - également, plusieurs valeurs possibles
    - 30 images/seconde, ...
- Principe du traitement du flux vidéo :
  - Succession d'images ! (« frame »)
  - Vérifier si une nouvelle frame est disponible pour récupérer l'image correspondante
  - Image = tableau de Pixels

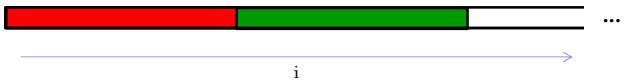
47

IV.1 – Format d'image vidéo sur webCam

- Format d'une image vidéo :
  - « Tableau » de pixels, ...
  - stocké sous la forme d'un vecteur de taille `numPixels` (valant `width x height`)



- Contenu effectif du tableau `webCam.pixels[i]`

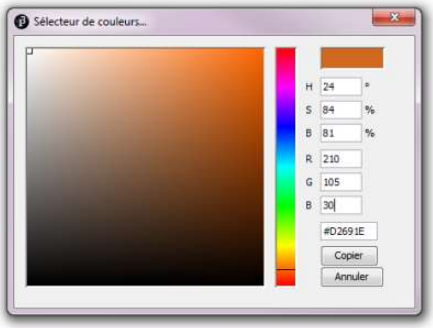


48

IV.1 – Format et couleur du pixel

- Format d'un pixel, sous Processing :
  - Couleur (nombre entier `int` ou `color`) sur 24 bits
  - 3 x 8 bits, donc (255x255x255 couleurs possibles)

  - ou encore : 4 x 8 bits, avec une composante Alpha (transparence)
  - Deux modes de gestion des couleurs :
    - RGB : « Red », « Green », « Blue »
    - HSB : « Hue », « Saturation », « Brightness »
- Sélecteur de couleurs (menu « outils ») :



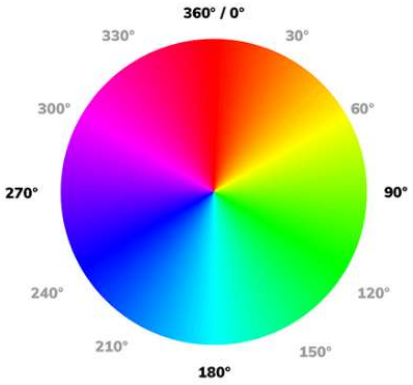
- Exemple en RGB :

```
colorMode(RGB,255,255,255);
chocolate = color(210,105,30);
// equivalent a :     chocolate = color(0xD2,0x69,0x1E);
//                    chocolate = #D2691E;
//                    chocolate = 0xFFD2691E; // 0xFF composante Alpha
```

49

IV.1 – Format et couleur du pixel

- Roue « chromatique » :
  - « Hue » (H, Teinte) : de 0 à 360°
  - « Saturation » (S) : 0 à 100%
  - « Brightness » (B, luminosité) : 0 à 100%



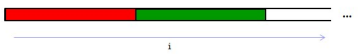
- Exemple en HSB :

```
chocolate = color(210,105,30); // en RGB
colorMode(HSB, 360, 100, 100); // changement de mode
float teinte = hue(chocolate); // ici, teinte = 25.00
```

50

IV.1 – Analyse de l'image en provenance de la webCam

- Traitement d'image = analyse des pixels
- Deux modes de parcours : vecteur (1D) ou matrice (2D)
  - Parcours sur le vecteur (index i) :

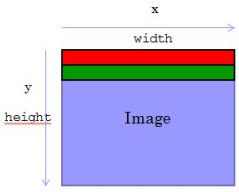



```
int i;            // index
color currColor; // Couleur du pixel courant
for (i = 0; i < (webCam.height*webCam.width); i++) {
  // Recuperation de la couleur pour chaque pixel...
  currColor = webCam.pixels[i];
  // Traitements sur les pixels...

}
```

  - Parcours sur la matrice (index x et y) :

```
int i, x, y, yPos;
for (y = 0; y < webCam.height; y++) { // lignes y
  yPos = y * webCam.width; // adresse debut de ligne
  for (x = 0; x < webCam.width; x++) { // colonnes x
    i = x + yPos;
    // Recuperation de la couleur pour chaque pixel..
    currColor = webCam.pixels[i];
    // Traitements sur les pixels...
  }
}
```



51

IV.2 – Bibliothèque standard de gestion WebCam : « Video »

- Bibliothèque « video » basée sur « Gstreamer »
  - A ajouter explicitement : menu Sketch > Importer une librairie... > Ajouter une librairie
  - puis chercher « Video » dans le « contribution manager » :

- Intégration de cette bibliothèque dans un programme Processing :
  - Import de la bibliothèque, à faire **avant** setup() :

```
import processing.video.*;
```

- Dans setup() : Ouverture de la webCam, avec les bons paramètres
  - Recherche d'une webCam, par interrogation du système d'exploitation de la machine :

```
String[] cameras = Capture.list();
```

  - Ouverture de la webCam, que si la recherche précédente a abouti

```
if (0 == cameras.length) {// pas de webCam... } else {
webCam = new Capture(this, 640, 480, cameras[0], 30);
```

  - Mise en marche effective de la webCam, pour création d'images à la fréquence souhaitée

```
webCam.start(); }
```

52

IV.2 – Bibliothèque standard de gestion WebCam : « Video »

- Utilisation du flux vidéo
  - Uniquement par des actions dans la fonction « Draw() » !
- Lecture d'une image sur le flux vidéo
  - Vérification au préalable de la présence effective d'une nouvelle « frame »

```
if (webCam.available()) {
```

  - Récupération du vecteur de pixels :

```
webCam.read(); }
```

- Traitement des images
  - Chargement du tableau de pixels

```
webCam.loadPixels(); // Conseille
```

  - Analyse et/ou modification du vecteur, donc, de l'image

```
... webCam.pixels[] ...
```

  - Mise à jour des pixels

```
webCam.updatePixels(); // Maintenant, obligatoire...
```

- Restitution (éventuelle) de l'image sur la fenêtre Processing

```
image(webCam, 0, 0);
```

53

IV.3 – Seconde possibilité : Bibliothèque Java « Sarxos »

- Bibliothèque externe à Processing
  - A ajouter **manuellement** car non disponible dans le « contribution manager »
  - Dézipper « Webcam_Sarxos » directement dans C:\Users\xxx\Documents\Processing\libraries
- Intégration de cette bibliothèque dans un programme Processing :
  - Import de la bibliothèque, à faire **avant** setup() :

```
import com.github.sarxos.webcam.*;    // Bibliotheque SARXOS de gestion webcam
import java.awt.image.BufferedImage; // Biblio pour conversion BufferedImage
import java.awt.Dimension;           // en PImage, a la bonne taille
import java.util.List;    // Pour recuperer la liste des webcams disponibles
```

- Dans setup() : Ouverture de la webCam, avec les bons paramètres
  - Recherche d'une webCam, par interrogation du système d'exploitation de la machine :

```
List<Webcam> cameras = Webcam.getWebcams();
```

  - Ouverture de la webCam, que si la recherche précédente a aboutie

```
if (cameras.isEmpty()) {// pas de webCam... } else {
webCam = Webcam.getDefault(); // Recuperation de la camera par defaut
webCam.setViewSize(new Dimension(640, 480)); // Choix de la resolution
PImgWebCam = createImage(640, 480, ARGB); // Creation de l'image de reception
```

  - Mise en marche effective de la webCam, pour création d'images à la fréquence souhaitée

```
webCam.open(); }
```

54

IV.3 – Seconde possibilité : Bibliothèque Java « Sarxos »

- Utilisation du flux vidéo
  - Uniquement par des actions dans la fonction « Draw() » !
- Lecture d'une image sur le flux vidéo
  - Vérification au préalable de la présence effective d'une nouvelle « frame »

```
if (webCam.isImageNew() && webCam.isOpen()) {
```

  - Récupération du vecteur de pixels, au format « BufferedImage » de Java :

```
BImgWebCam = webCam.getImage(); }
```

- Traitement des images
  - Conversion en PImage, format standard de Processing

```
BImgWebCam.getRGB(0, 0, 640, 480, PImgWebCam.pixels, 0, 640);
PImgWebCam.updatePixels();
```

  - Analyse et/ou modification du vecteur, donc, de l'image

```
... PImgWebCam.pixels[] ...
```

  - Mise à jour des pixels

```
PImgWebCam.updatePixels(); // Maintenant, obligatoire...
```

- Restitution (éventuelle) de l'image sur la fenêtre Processing

```
image(PImgWebCam, 0, 0);
```

55

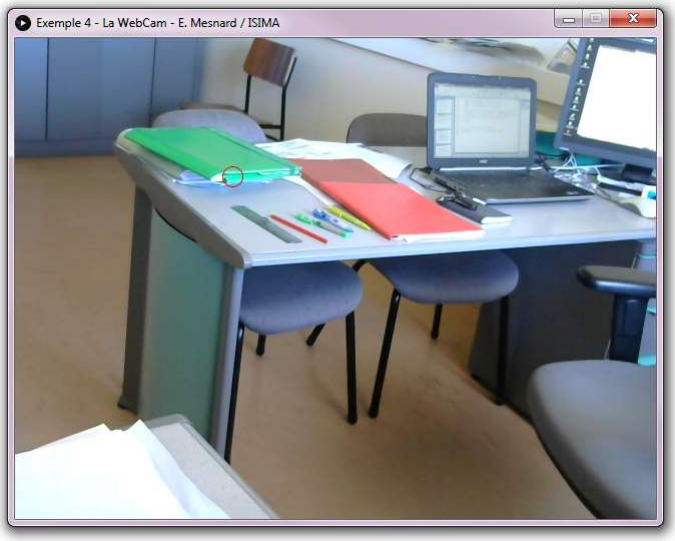
IV.4 – Exemple 4 : recherche d'un point vert sur webCam

```
    // Recherche du point le plus proche de la couleur de reference
    poidsPOI = 360; // Valeur la plus grande possible
    xPOI = 0; yPOI = 0;   // Par defaut, le POI est en (0,0)
    // Analyse de l'image
    for (yy = 0; yy < heightCapture; yy++) { // abscisse yy
      yPos = yy * widthCapture;
      for (xx = 0; xx < widthCapture; xx++) { // ordonnees xx
        i = xx + yPos;
        currColor = webCam.pixels[i]; // recuperation couleur
        teinte = hue(currColor);      // et de la teinte

        // Calcul de l'ecart de teinte par rapport au vert pur
        poids = abs(120-teinte); // car vert pur = 120 degre
        if (poids < poidsPOI) {  // Le POI est le point qui a le moins de difference...
          poidsPOI = poids; // Mise a jour des informations et coordonnees
          xPOI = xx;
          yPOI = yy;
        }
      }
    } // A la sortie de cette boucle, le POI est (xPOI,yPOI)
    image(webCam, 0, 0); // Restitution de l'image captee sur la webCam
    ellipse(xPOI,yPOI,20,20); // Trace d'un cercle rouge autour du POI
  }
}
```

56

IV.4 – Exemple 4 : Résultat d'exécution !



57

TD 2 : Traitement d'image

1. Faire la recherche du point vert avec la bibliothèque « Sarxos »
2. Application simpliste de « traitement d'image » : miroir horizontal avec « Sarxos » et/ou « Vidéo »

- Déclarer un objet « image » pour contenir l'image inversée :
```
PImage webCamMirror;
```
- Créer le tableau de pixels associé
```
webCamMirror = createImage(640,480,RGB);
```
- Gérer l'inversion (dans la fonction `draw()`)
- Afficher le résultat sur la fenêtre, en plein écran :
```
image(webCamMirror,0,0);
```

- Eventuellement :
  - Créer des constantes,
  - Gérer le clavier pour proposer l'image normale ou inversée (touche M)

58