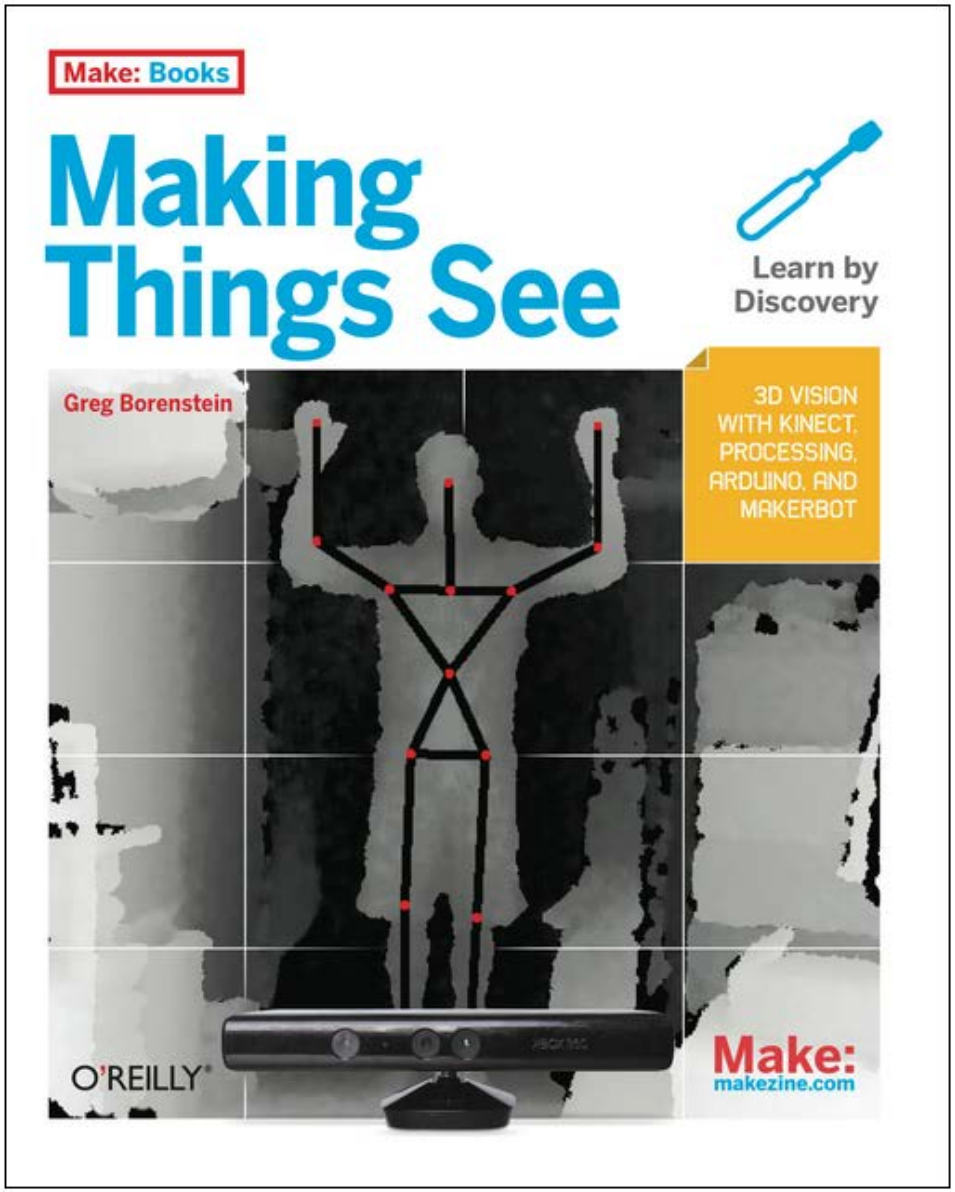
UTILISATION DE LA KINECT

… et l’ordinateur vit !!!

Bibliographie:

Site:

- http://makingthingssee.com/



Pdf: (entre autres…)

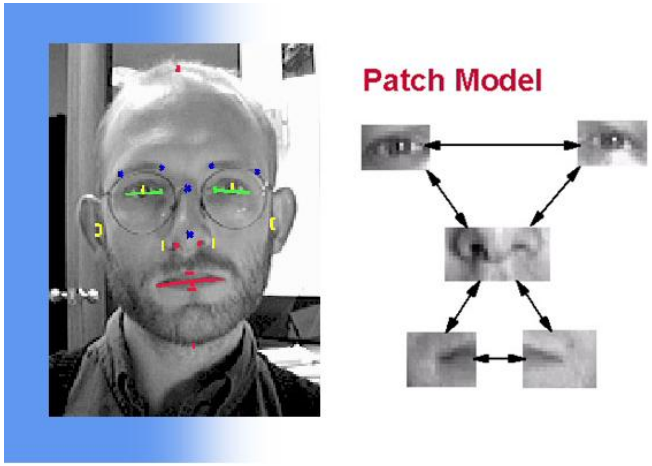
- http://csce.uark.edu/~jgauch/5703/other/books/Making_Things_See.pdf

HISTORIQUE ET POSSIBILITÉS DE LA KINECT

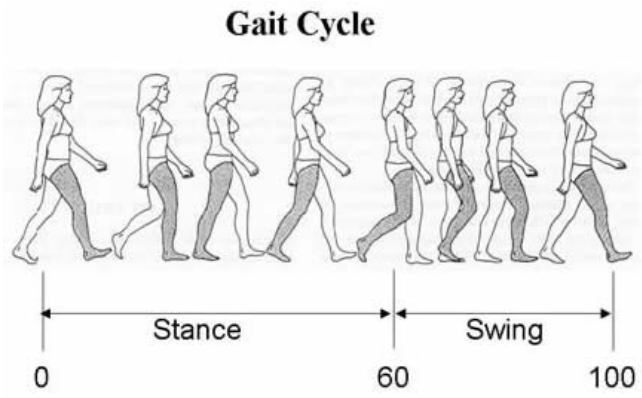
- La kinect est le fruit de la recherche militaire développée pour détecter les terroristes dans l'espace public…

- Les technologies:

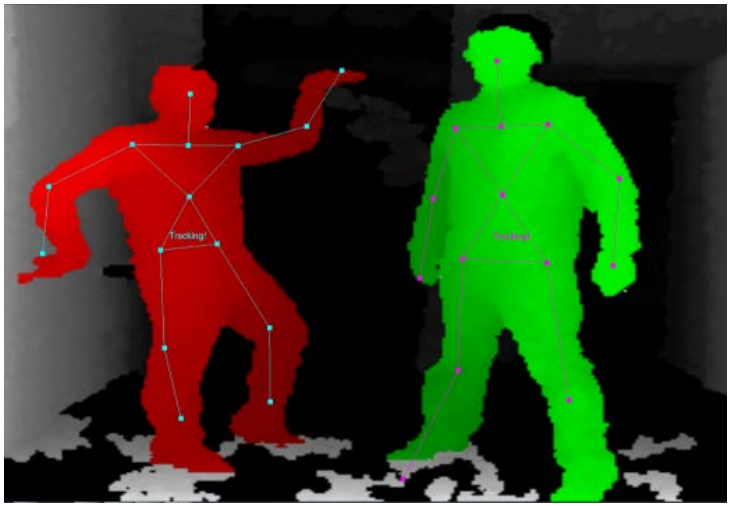
Reconnaissance faciale



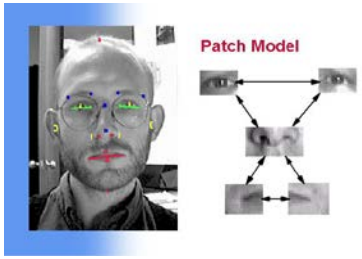
Analyse de démarche

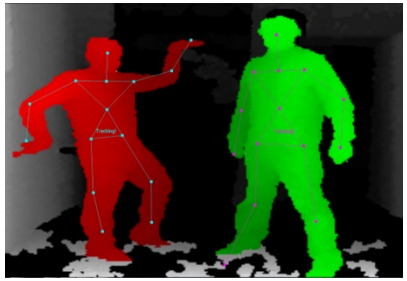


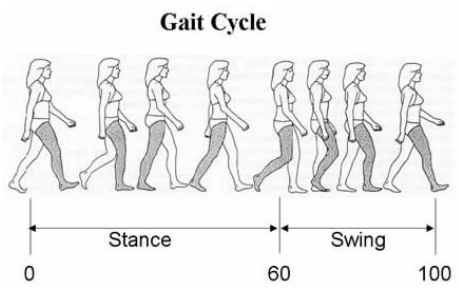
Squelettisation (du corps)



Imagerie en profondeur (3D)







Toutes ces technos

Espace public
(projets créatifs à but civil)

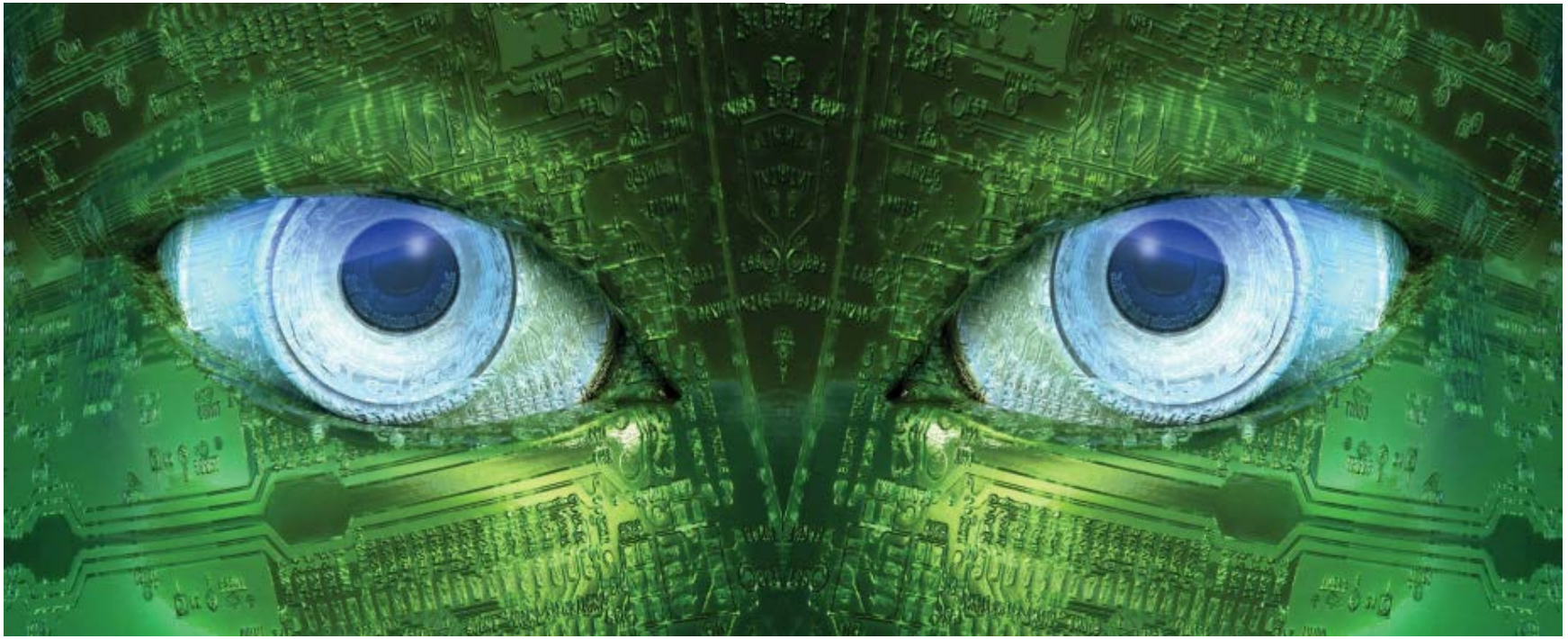
Interfaces gestuelles de logiciels

Scaner 3D

Motion capture

Etc…

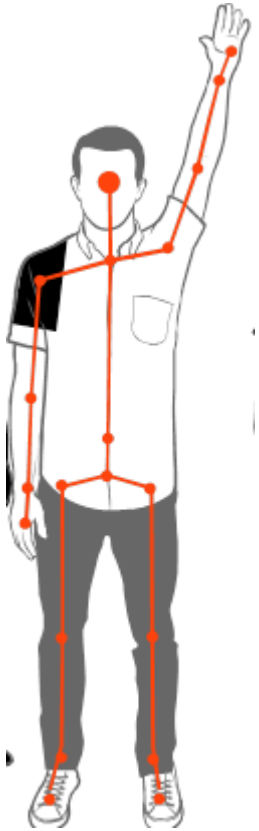
Les applications sont vastes !!



L’ordinateur peut voir en 3D !

Les possibilités de la kinect

- Travailler sur les information de profondeur issues de la camera 3D.
- Analyser et manipuler un nuage de points



- Traquer le mouvement des articulation du corps
- Détection de la gestuelle et des poses
- Analyse de scène et suppression du fond…

Structure du tutoriel

1. What Is the Kinect? 1

Qu'est-ce que la kinect ?

2. Working with the Depth Image 43

Travailler sur l'image en profondeur

3. Working with Point Clouds 109

Travailler avec les nuages de points

4. Working with the Skeleton Data 185

Travailler avec les données du squelette (articulations du corps)

5. Scanning for Fabrication 301

Fabrication d'un scanner 3D

CHAPITRE 1

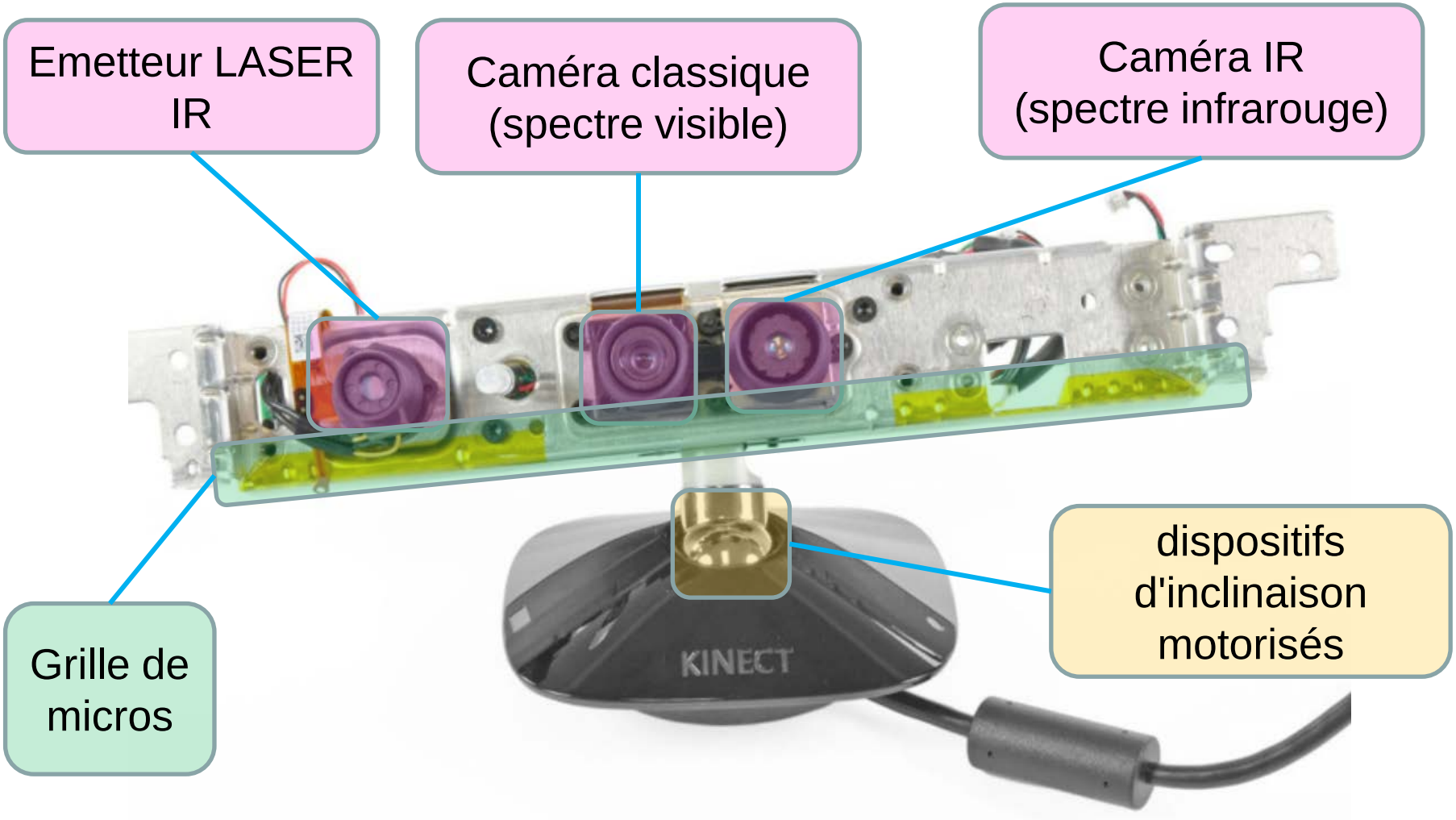
QU'EST-CE QUE LA KINECT ?

I. QU'EST-CE QUE LA KINECT ?

1.1. Architecture et fonctionnement de la kinect

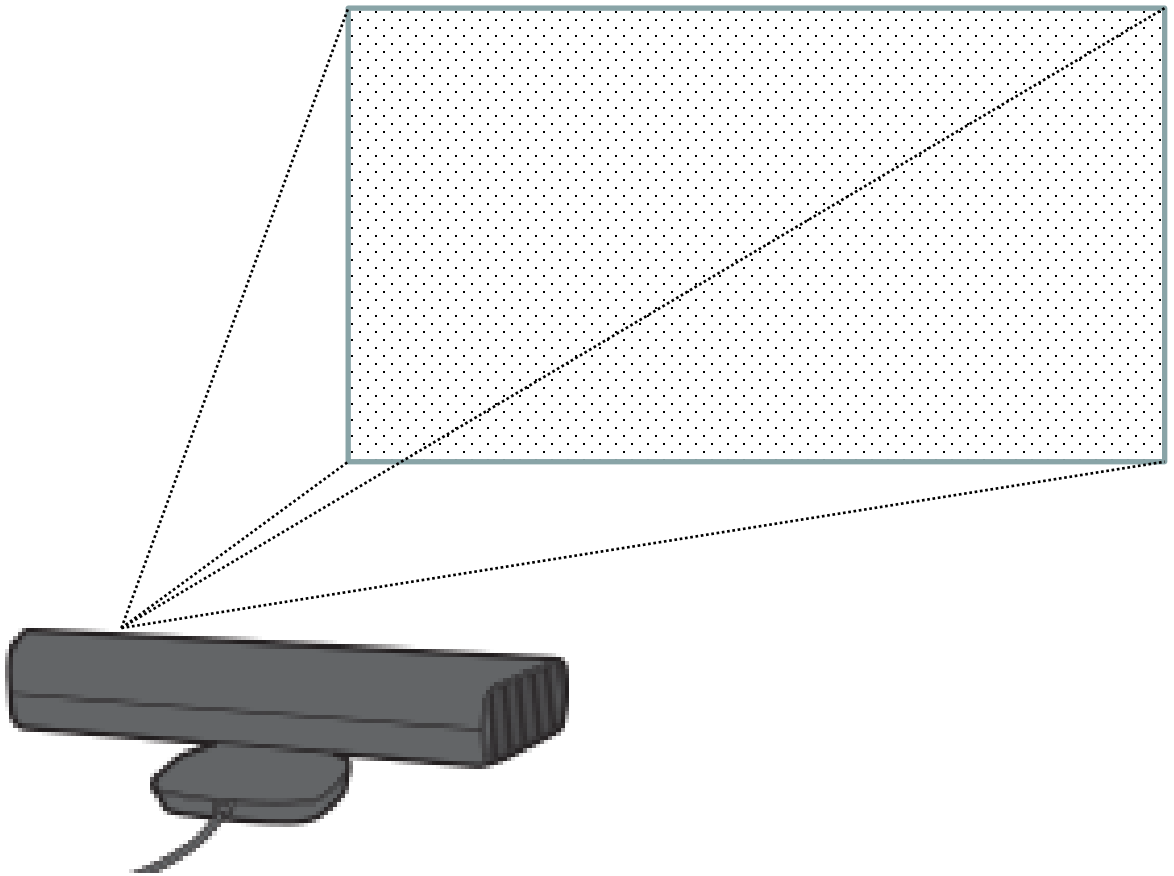
Architecture

En apparence: 3 yeux…

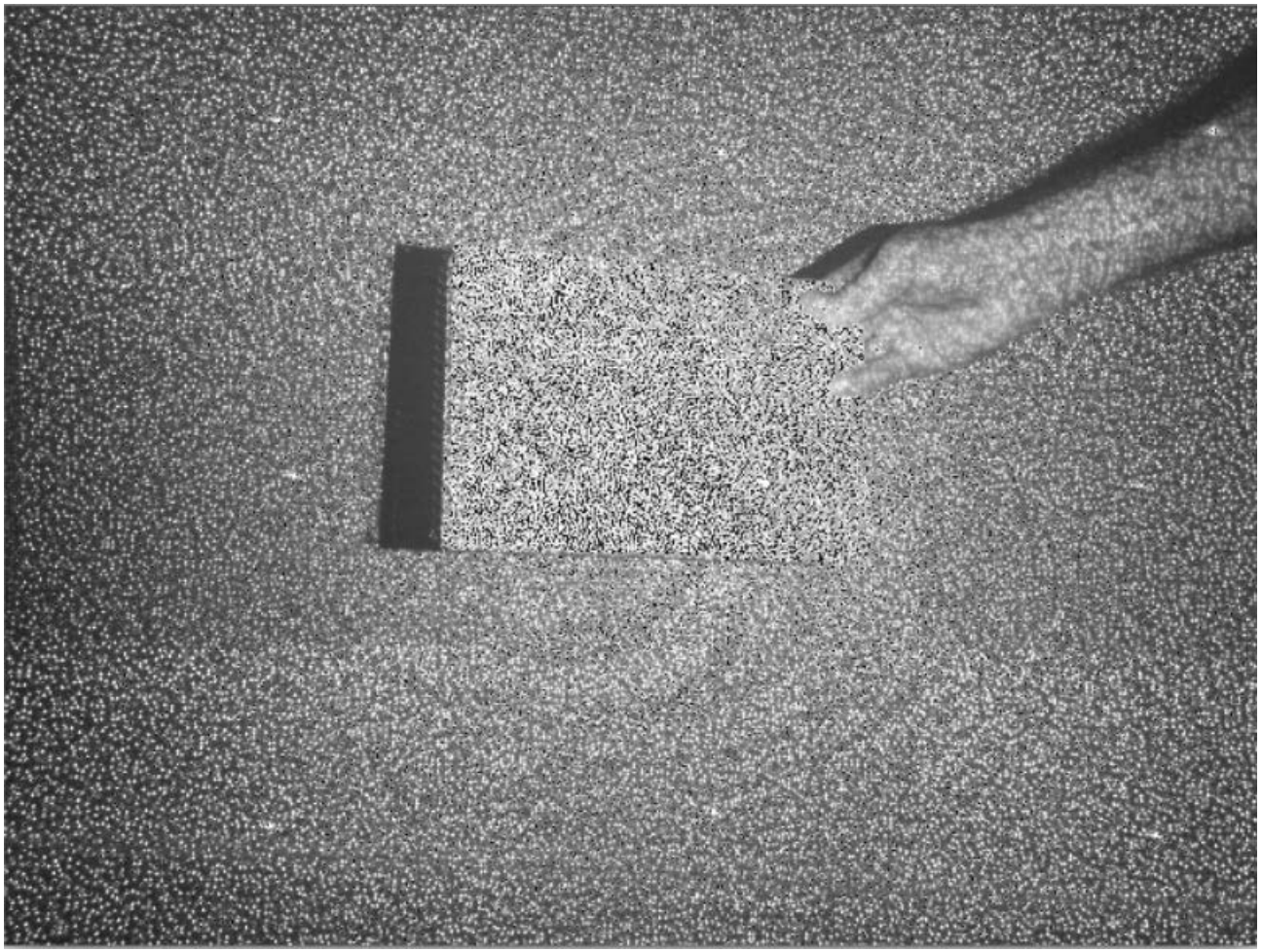


Secret de la kinect: Projecteur/Camera IR

- Projection d'une grille de points lumineux IR sur la scène (spekel) …

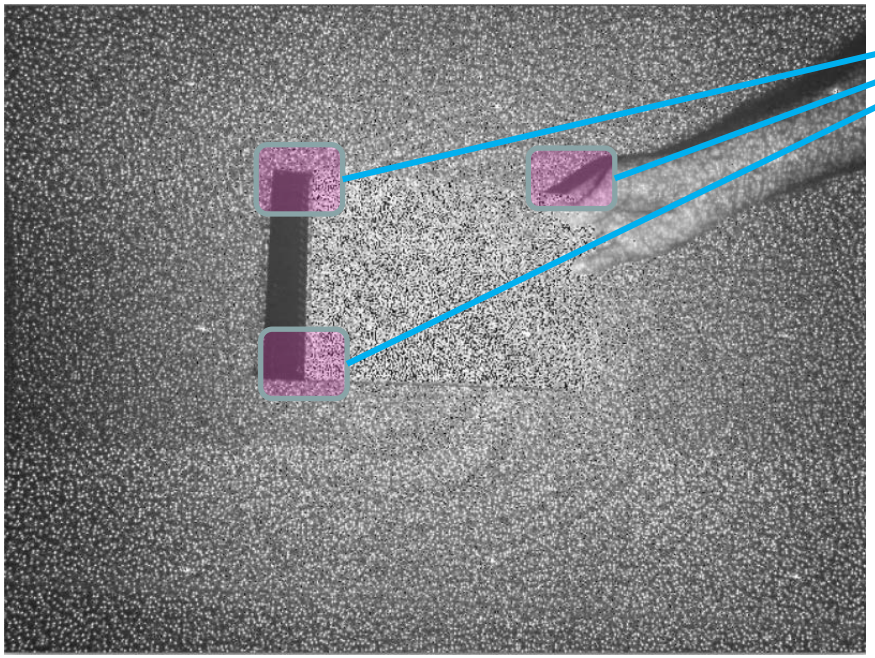


- On obtient l’image suivante:



- Cette image est comparée à une image de référence étalonnée par le fabriquant sur un mur plan situé à une distance connue

L’estimation de la distance est effectuée en calculant le déplacement d’un point par rapport à la grille originale



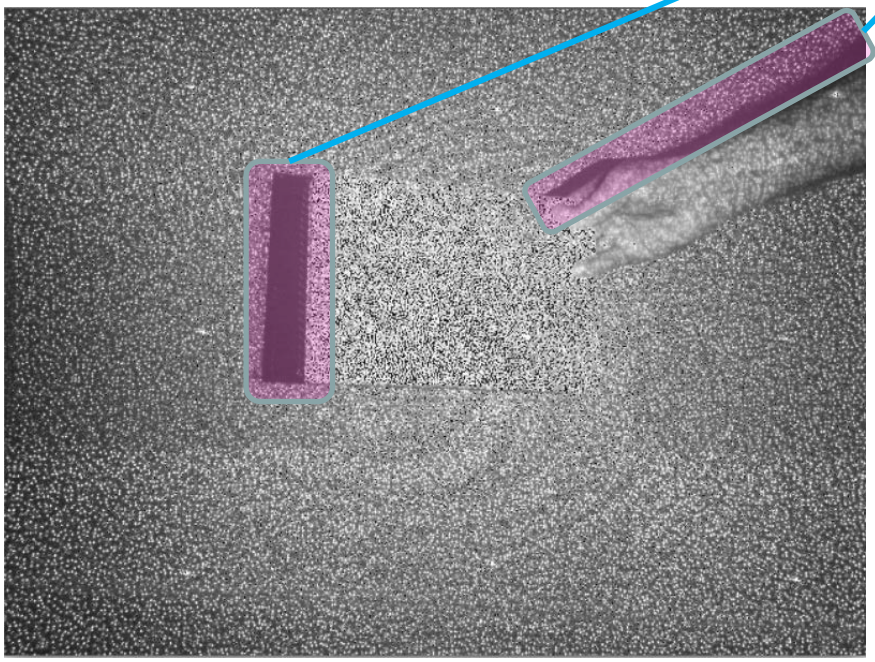
Déplacement des points % grille originale

la kinect est calibrée pour connaitre la position originelle de chacun des points

Elle utilise leur déplacement pour estimer la **distance d’un objet** dans cette partie de la scène

Cette estimation est la base de toutes les possibilités de la kinect…

Les limitations…



Ombres projetées par les objets touchés par la lumière IR

Aucun point de la projection infrarouge effectuée par la kinect n'atteint **les parties ombragées**

La kinect est incapable d'estimer la distance des points situés dans cette partie de l'espace

Les autres matériels…

Webcam basse résolution classique (640*480 pix)

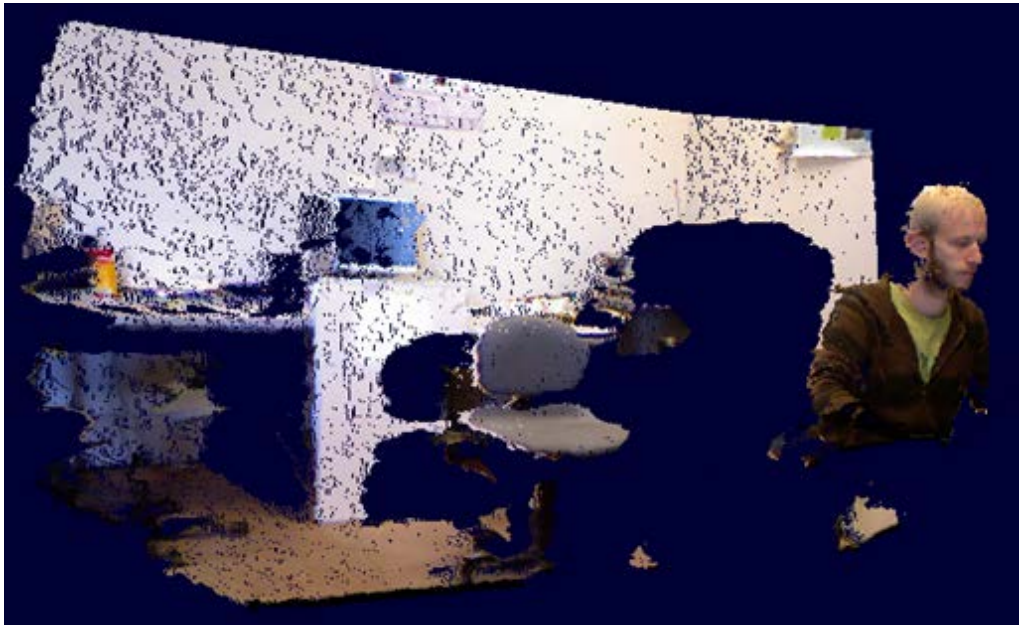
Possibilité de superposer l'image en couleur sur l'image en profondeur

Colorier l'image 3D

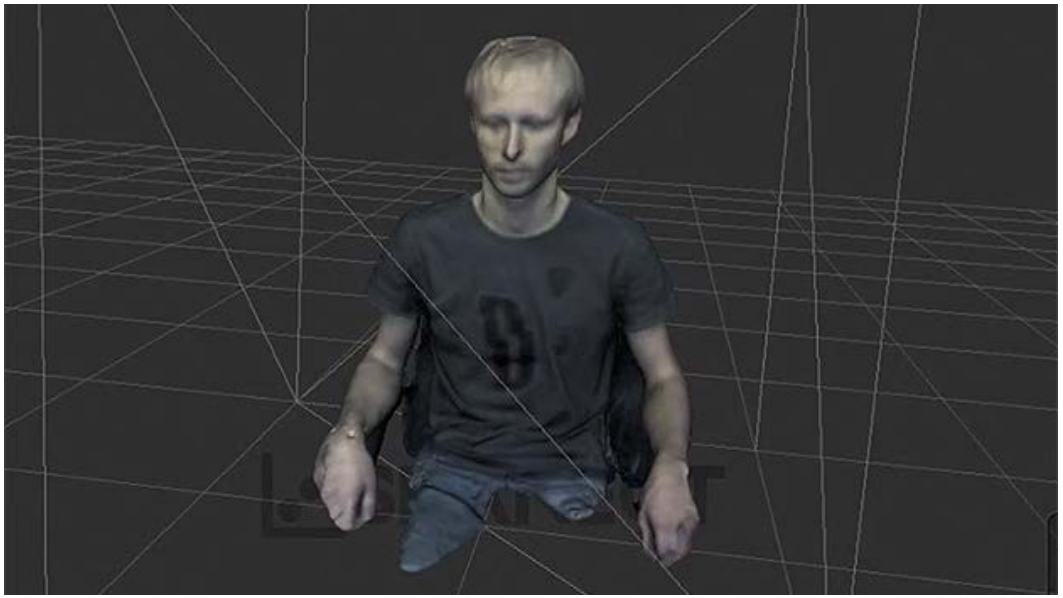
Créer des scan 3D prenant en compte la couleur réelle des objets

Exemples:

Image en profondeur colorée



Scan 3D coloré

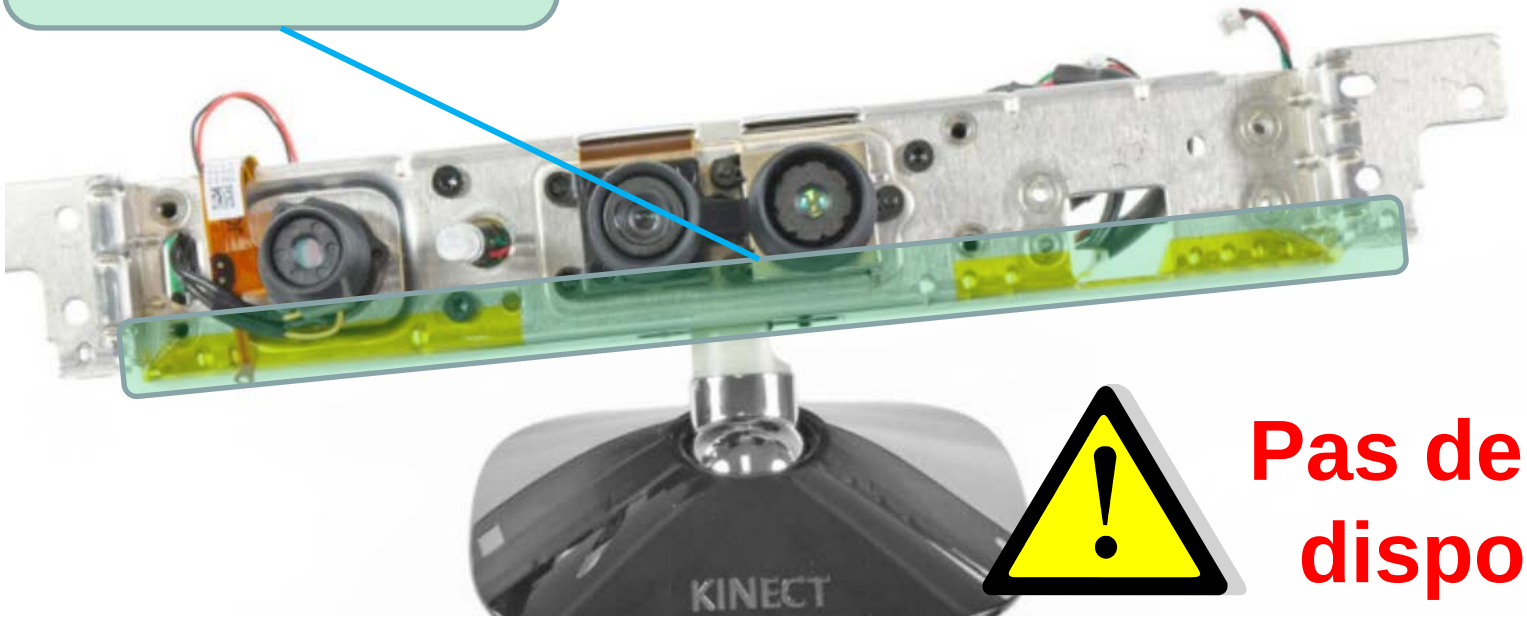


Les autres matériels…

Possibilité de localiser les différents joueurs dans l'espace…

Attribution d'une commande particulière à un joueur particulier

4 micros distribués le long de la kinect



Pas de librairie disponible !!

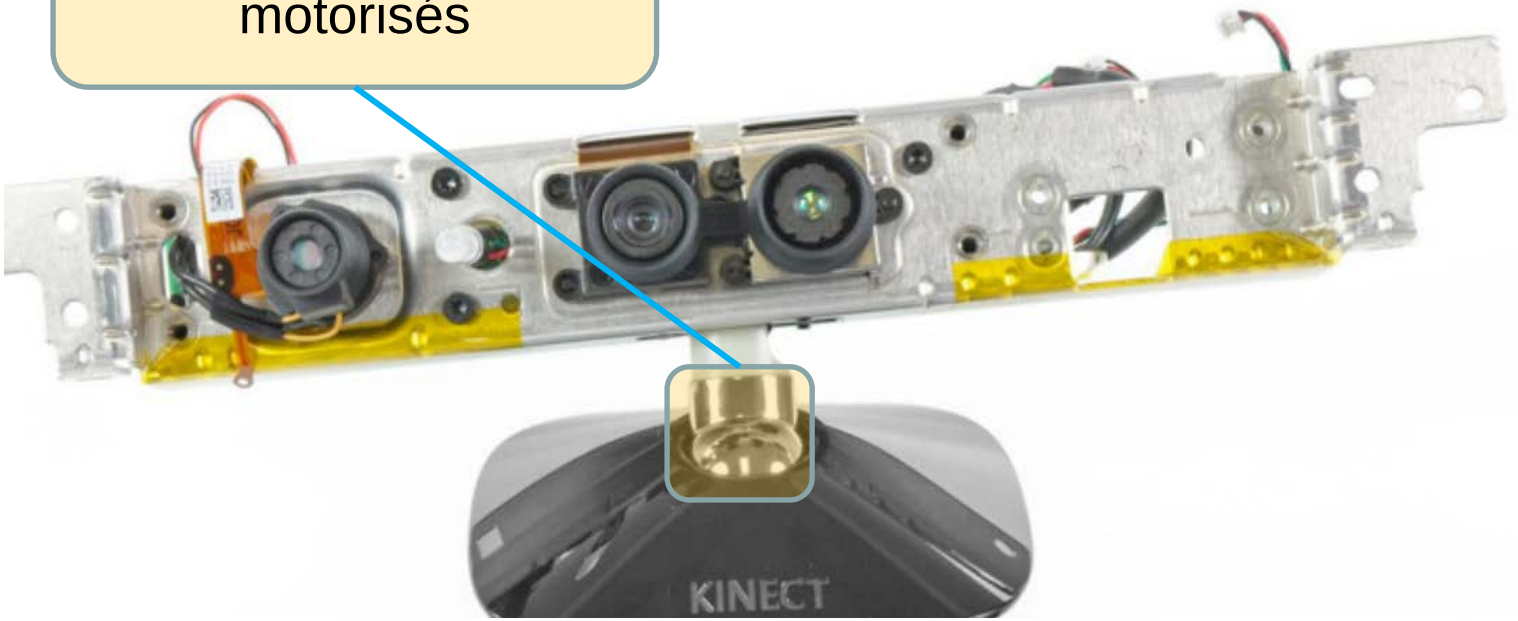
Les autres matériels…

Possibilité de rotation de haut en bas de la caméra et des micros **(30°)**

Adaptation à n'importe quel type de salle et nombre de joueur

Calcul de la meilleur position pour l'enregistrement des différents joueurs

dispositifs d'inclinaison motorisés



1.2. Intérêt de la kinect par rapport à une caméra classique

⇨ L'image en profondeur est plus facile à analyser pour la machine qu'une image en couleur classique.

Image couleur classique:

↳ La reconnaissance d'objets et de personne passe par l'analyse des pixels…

↳ Valeur des pixels dépend des conditions de prise de vue : éclairage, ouverture, modification des couleurs par l'appareil, etc…

⇨ Impossibilité de connaitre facilement le contenu d'une image, de savoir où commence et où finit un objet, etc…

Image en profondeur:

- L'intensité des pixels est proportionnelle à la distance de l'objet considéré avec la caméra.
- L'intensité des pixels est indépendante de l'éclairage

⇒ possibilité de localiser avec précision les objets dans la scène, où ils commencent / finissent, etc…

⇒ Reconstruction 3D de la scène.

⇒ Image en profondeur permet d'appliquer des algorithmes de tracking plus efficace que sur une image classique:

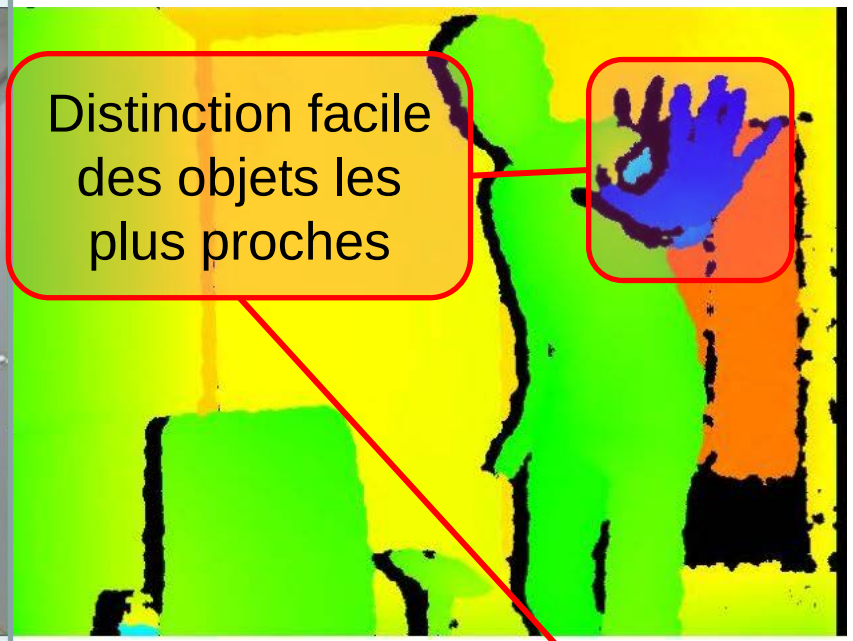
- Tracking des personnes
- Tracking des articulations du corps

} Librairies performantes…

Exemples:

Image en couleur classique

Image en profondeur



1.3. Création de la kinect et hacking opensource

- Kinect développé par Microsoft – périphérique de la Xbox 360…
- Résultat d'années de recherche académique dans l'imagerie robotique (computer vision)…
- La hardware a été développé par PrimeSense, une entreprise israélienne qui a d'autres caméra en profondeur basées sur la technique de projection IR…
- 4 novembre 2010 : lancement de la kinect (association des algorithme de Microsoft & hardware PrimeSense)
  - Succès commercial : 10M d'unité vendu le 1er mois !!
  - Début du hacking : développement des drivers opensource.. (kinect = système propriétaire fermé!)
- Adafruit (compagnie NY): 3000$ de récompense pour 1er driver opensource

- Adafruit (compagnie NY): 3000$ de récompense pour 1er driver opensource
  - Le 10 novembre (1 semaine après le lancement !!) : 1er driver public qui permet de manipuler l'image en profondeur par Hector Martin.
  - Etablissement du projet « OpenKinect » qui propose jusqu'à aujourd'hui des améliorations…
  - Explosion de librairie pour différents environnement…
  - En particulier développement de librairies pour **Processing** (Dan Shiffman).
- **En réponse à cet intérêt PrimeSense Libère ses drivers de la kinect !!**
  - Système OpenNI : Natural Interaction

Deux systèmes opensource pour contrôler la kinect :

Choix de l'auteur

OpenKinect

- Accès aux servo-moteurs
- Licence « full opensource »
- Acces à l'image en profondeur »
- Pas de Tracking des articulations de plusieurs personnes

OpenNI

- Pas d' ccès aux servo-moteurs
- License GNU-LGPL (similaire à openkinect)
- Certains modules non-accessibles
- Acces à l'image en profondeur »
- Tracking des articulations de plusieurs personnes

CHAPITRE 2

TRAVAILLER SUR L'IMAGE EN PROFONDEUR

II. TRAVAILLER SUR L'IMAGE EN PROFONDEUR

2.1. Installation de la librairie SimpleOpenNI pour Processing

- En 2 phases:
  - Installation de la librairie OpenNI sur l'ordinateur
  - Installation de la librairie SimpleOpenNI sur Procesisng

Instructions sur:

https://code.google.com/p/simple-openni/wiki/Installation

- Mac : OS X → A priori, pas d'install préalable (voir site)
- Windows: télécharger et installer **Kinect SDK**

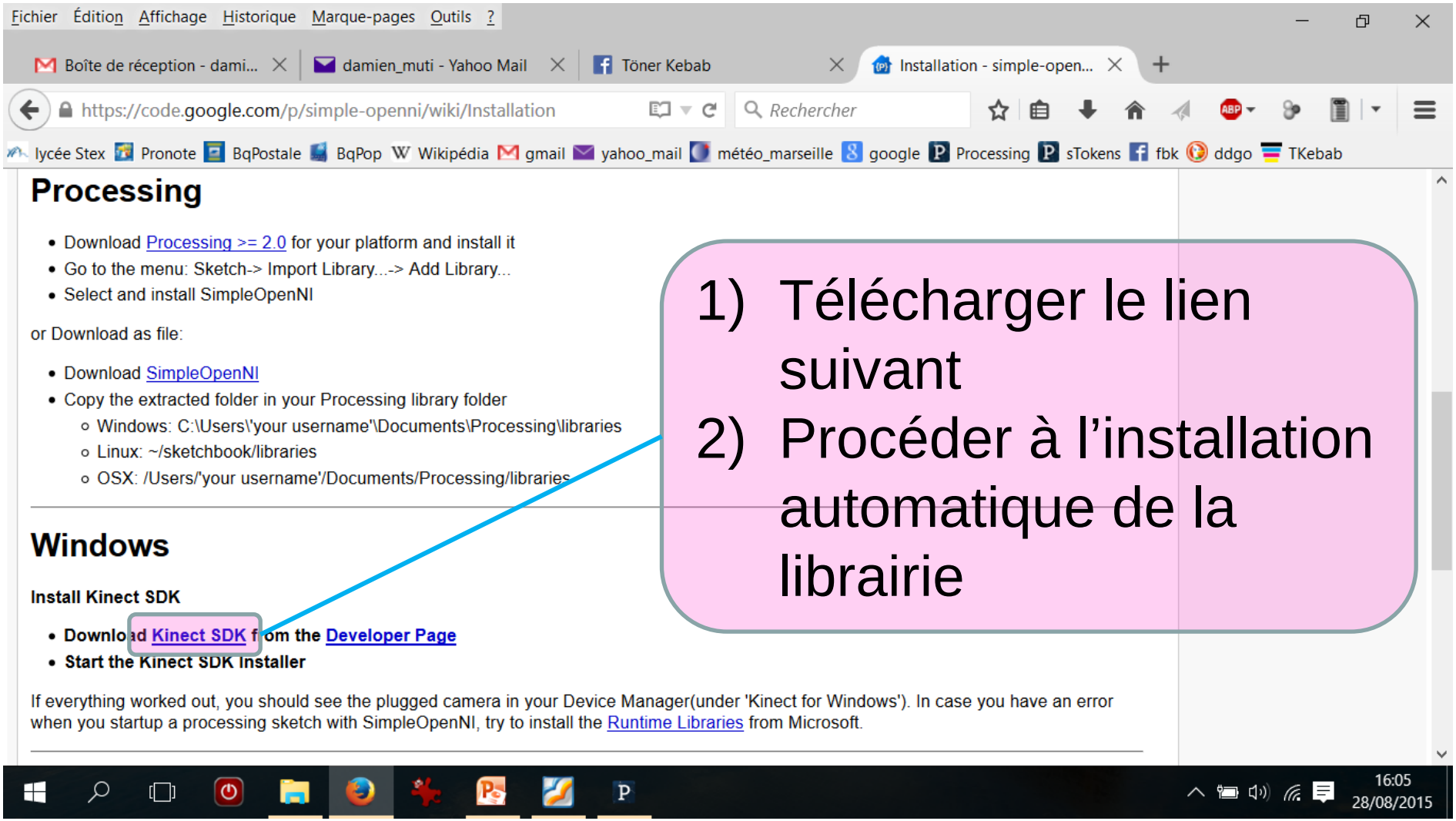
Installation rapide sur: À tester…

http://zigfu.com/ > « download plugin »

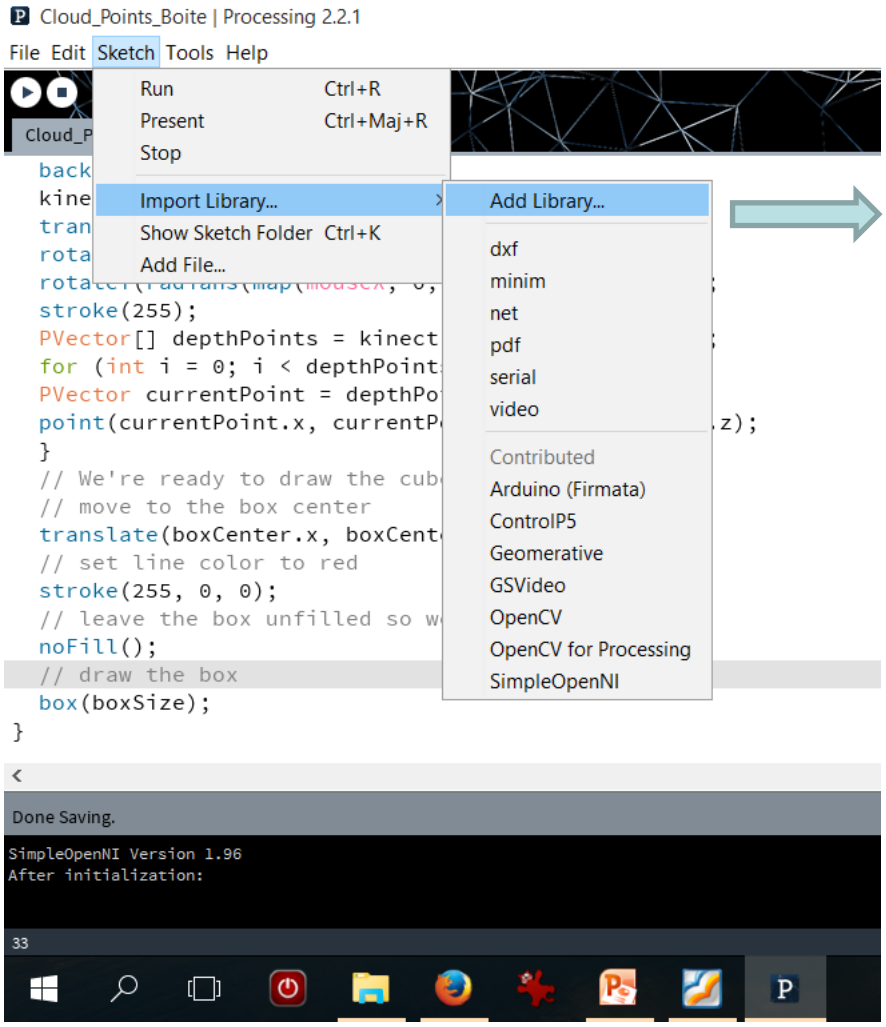
- Windows / MAC

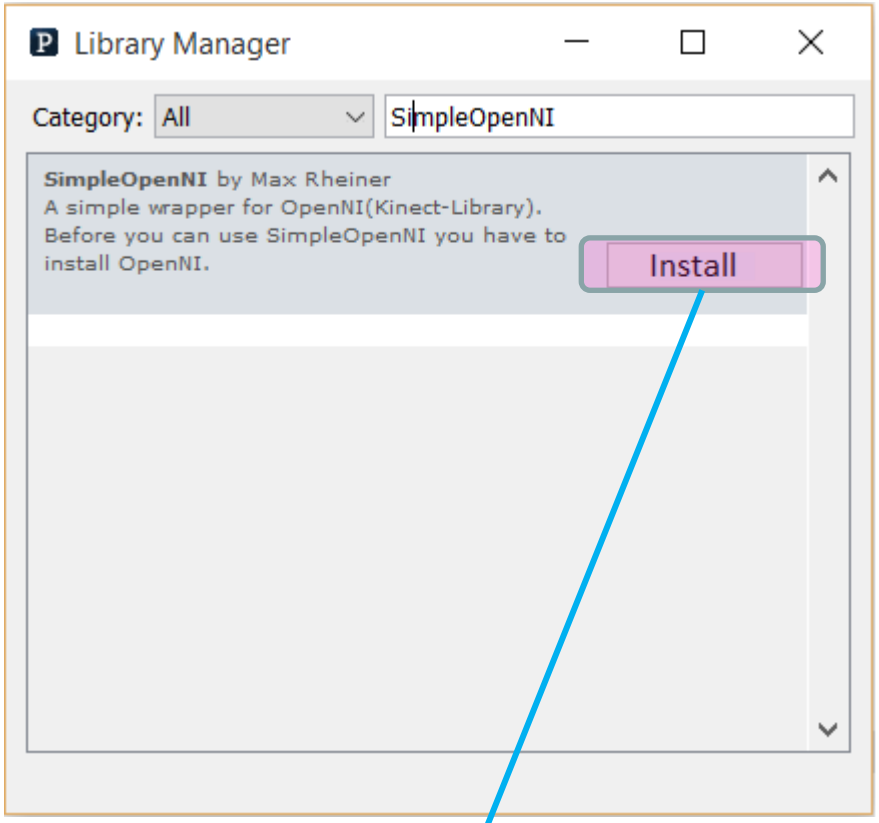
Windows:

https://code.google.com/p/simple-openni/wiki/Installation



Installation de la librairie SimpleOpenNI sous Processing

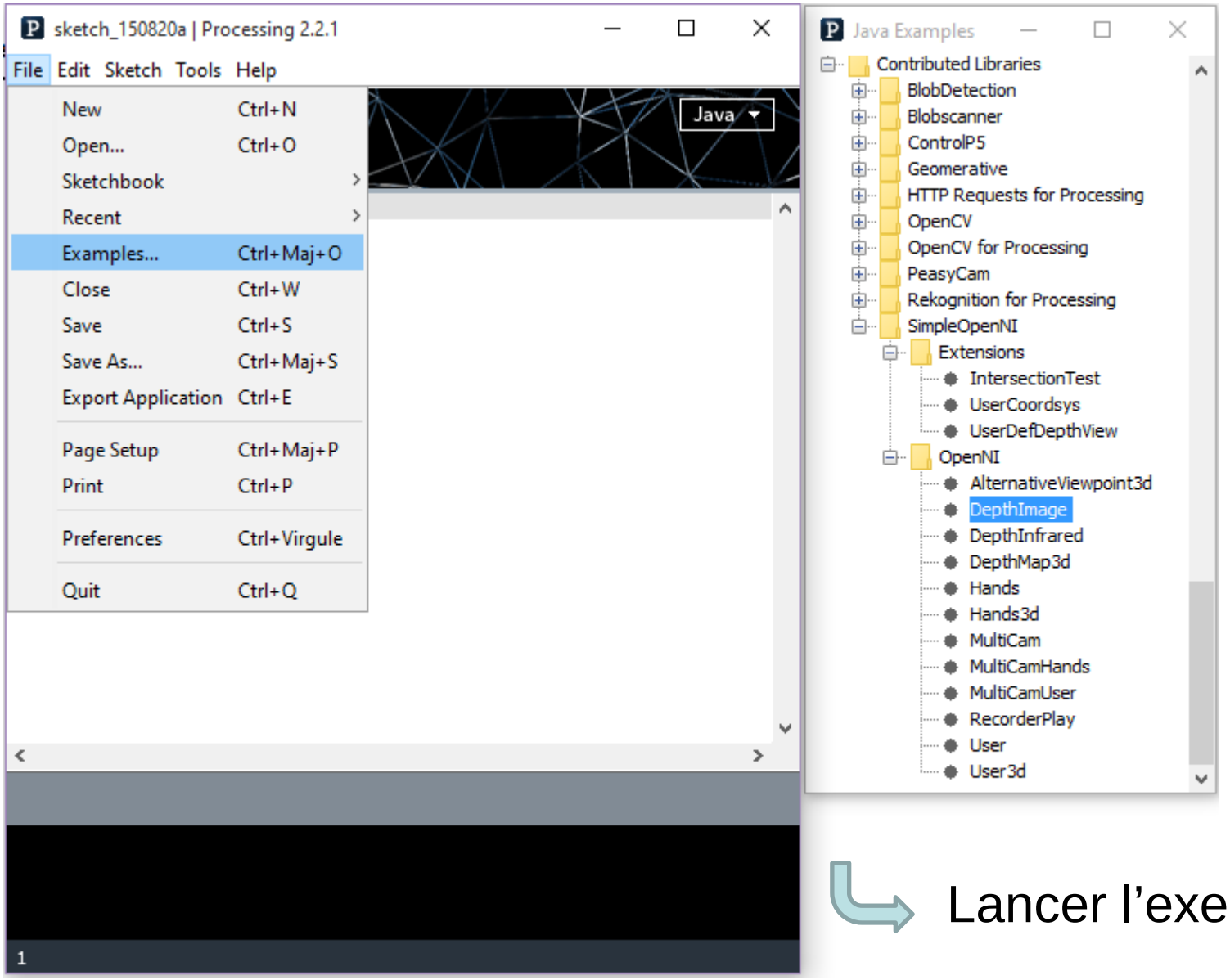




Lancer l'installation

1er test: sur Processing 2.2.1

File > Examples > contributed Libraries > SimpleOpenNI > OpenNI > DepthImage



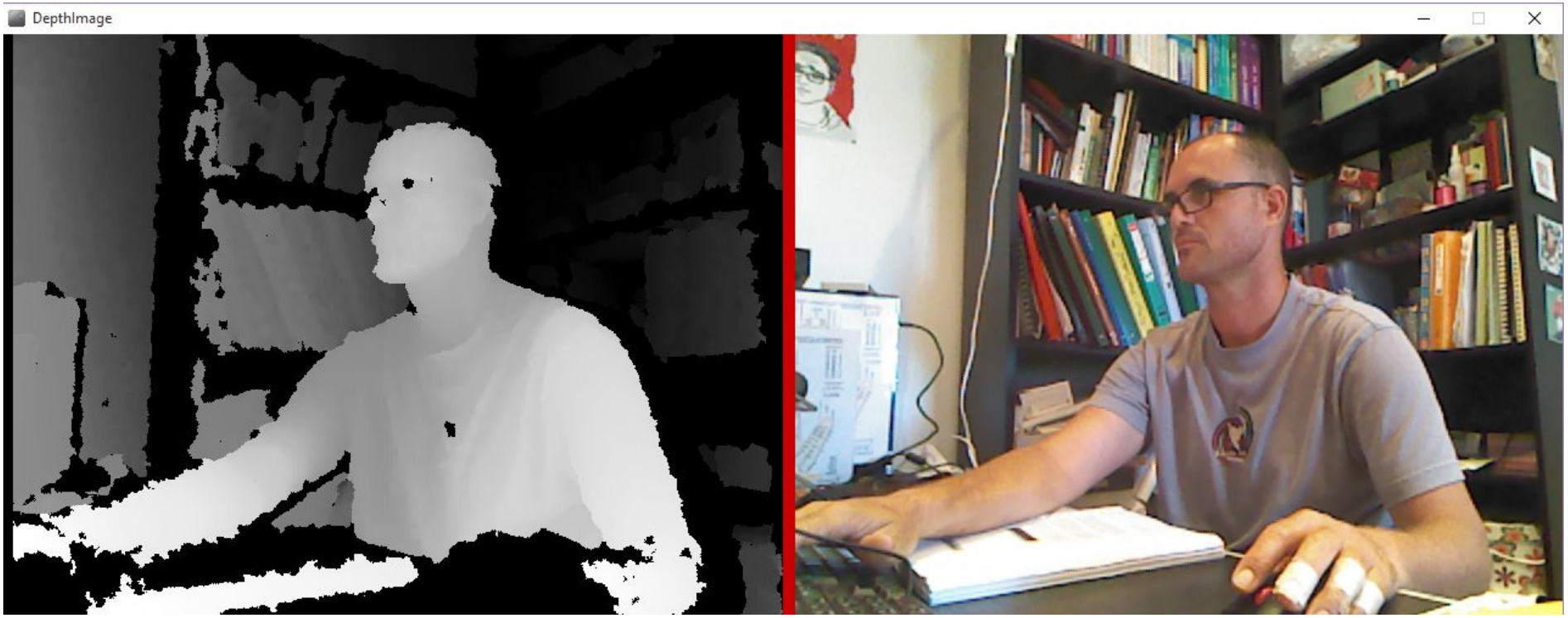
Lancer l'exemple…

Programme:

```
/* --------------------------------------------------------------------------
 * SimpleOpenNI DepthImage Test
 * --------------------------------------------------------------------------
 * Processing Wrapper for the OpenNI/Kinect 2 library
 * http://code.google.com/p/simple-openni
 * --------------------------------------------------------------------------
 * prog:  Max Rheiner / Interaction Design / Zhdk / http://iad.zhdk.ch/
 * date:  12/12/2012 (m/d/y)
 * --------------------------------------------------------------------------
 */

import SimpleOpenNI.*;


SimpleOpenNI  context;

void setup()
{
  size(640*2, 480);
  context = new SimpleOpenNI(this);
  if (context.isInit() == false)
  {
    println("Can't init SimpleOpenNI, maybe the camera is not connected!");
    exit();
    return;
  }

  // mirror is by default enabled
  context.setMirror(true);

  // enable depthMap generation
  context.enableDepth();

  // enable ir generation
  context.enableRGB();
```

```
void draw()
{
  // update the cam
  context.update();

  background(200, 0, 0);

  // draw depthImageMap
  image(context.depthImage(), 0, 0);

  // draw irImageMap
  image(context.rgbImage(), context.depthWidth() + 10, 0);
}
```

Si tout fonctionne, on obtient:



2.2. Commentaire du programme et analyse de la « depth Image » (DI)

```
import SimpleOpenNI.*;
```

Importation de la librairie « SimpleOpenNI »

```
SimpleOpenNI  context;
```

Déclaration d'un objet « context » héritée de la classe « SimpleOpenNI »

```
void setup()
{
  size(640*2, 480);
  context = new SimpleOpenNI(this);
  if (context.isInit() == false)
  {
    println("Can't init SimpleOpenNI, maybe the camera is not connected!");
    exit();
    return;
  }

  // mirror is by default enabled
  context.setMirror(true);

  // enable depthMap generation
  context.enableDepth();

  // enable ir generation
  context.enableRGB();
```

Initialisation de l'objet « context » par instanciation de la classe « SimpleOpenNI »

Ecriture d'un message d'erreur si la kinect n'est pas accessible…

2.2. Commentaire du programme et analyse de la « depth Image » (DI)

```
import SimpleOpenNI.*;


SimpleOpenNI  context;

void setup()
{
  size(640*2, 480);
  context = new SimpleOpenNI(this);
  if (context.isInit() == false)
  {
    println("Can't init SimpleOpenNI, maybe the camera is not connected!");
    exit();
    return;
  }

  // mirror is by default enabled
  context.setMirror(true);

  // enable depthMap generation
  context.enableDepth();

  // enable ir generation
  context.enableRGB();
```

Appel à la méthode setMiror de l'objet « context ». (à compléter)

La méthode « enableDepth » permet l' accès à l'image en profondeur

La méthode « enableRGB » permet l' accès à l'image en couleur

2.2. Commentaire du programme et analyse de la « depth Image » (DI)

```
void draw()
{
  // update the cam
  context.update();

  background(200, 0, 0);

  // draw depthImageMap
  image(context.depthImage(), 0, 0);

  // draw irImageMap
  image(context.rgbImage(), context.depthWidth() + 10, 0);
}
```

Rafraichissement des données issues de la kinect : RGB-image et DI-Image

Réglage d'un fond rouge (r,v,b)=(200,0,0)

Affichage de l'image en profondeur « depthImage » héritée de la classe PImage

Affichage de l'image en couleur « rgbImage » héritée de la classe PImage

Les limitations de l'image en profondeur (Depth Image)

Distance minimale de détection (minimum range)

~ 50 cm

Les zones trop proches sont en noire, i.e. non détectées !!

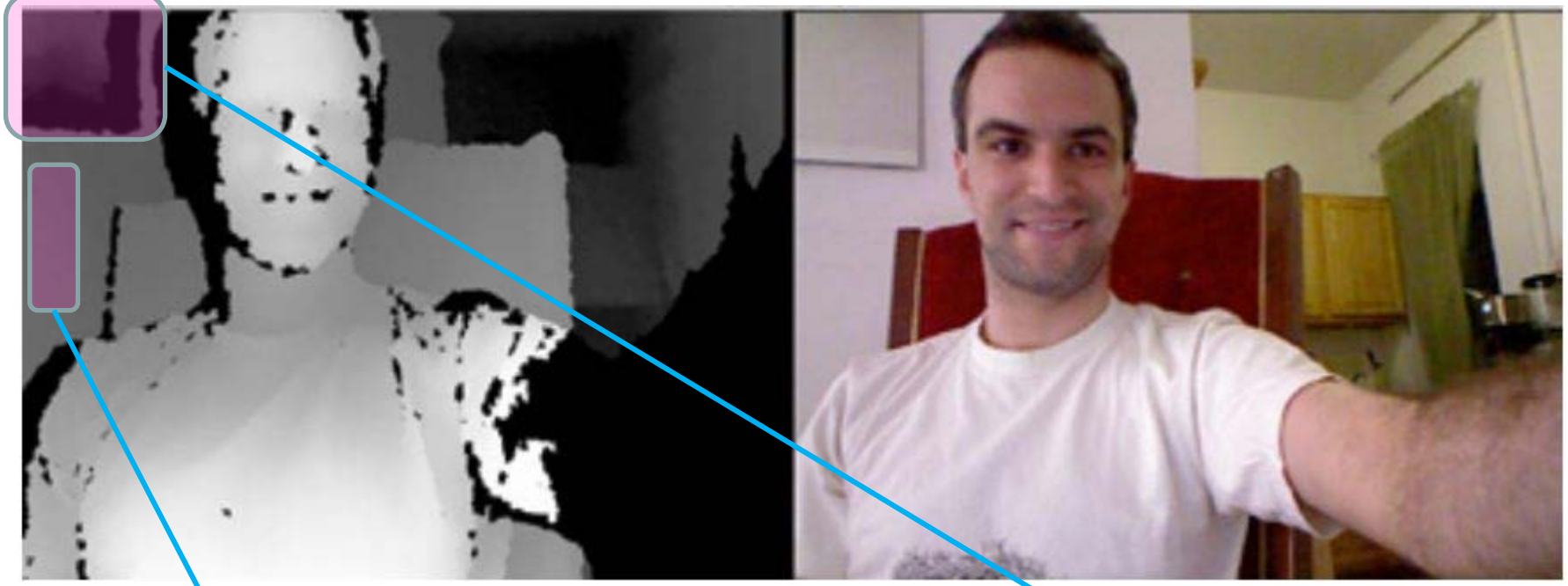
Bras très proche de la kinect

Bruit sur les contours des objets

Zones noires autour des contours des objets – frontières entre objets.

- Mesure de la distance uniquement si la lumière des points émis par le projecteur IR est réfléchie vers le récepteur IR.
- La réflexion de la lumière IR aux frontières s'effectue suivant des angles qui ne permettent pas le retour vers le récepteur IR
- Trou dans les données de la kinect → pixels noirs

Réflexions sur les surfaces lisses (miroirs, vitres, etc…)



Le mur est d'un certain niveau de gris

Le miroir sur le mur (à la même distance !!) a un niveau de gris différent de celui du mur et non uniforme !!

- La lumière IR est réfléchie par la surface du miroir
  - Pas de diffusion de la lumière IR qui permet le retour de celle-ci ver le récepteur IR
- Une partie de la lumière diffusée par les objets de la pièce touchés par les points IR se réfléchit sur le miroir et atteint le récepteur IR → distances perçues sont celles des images des objets par la miroir…

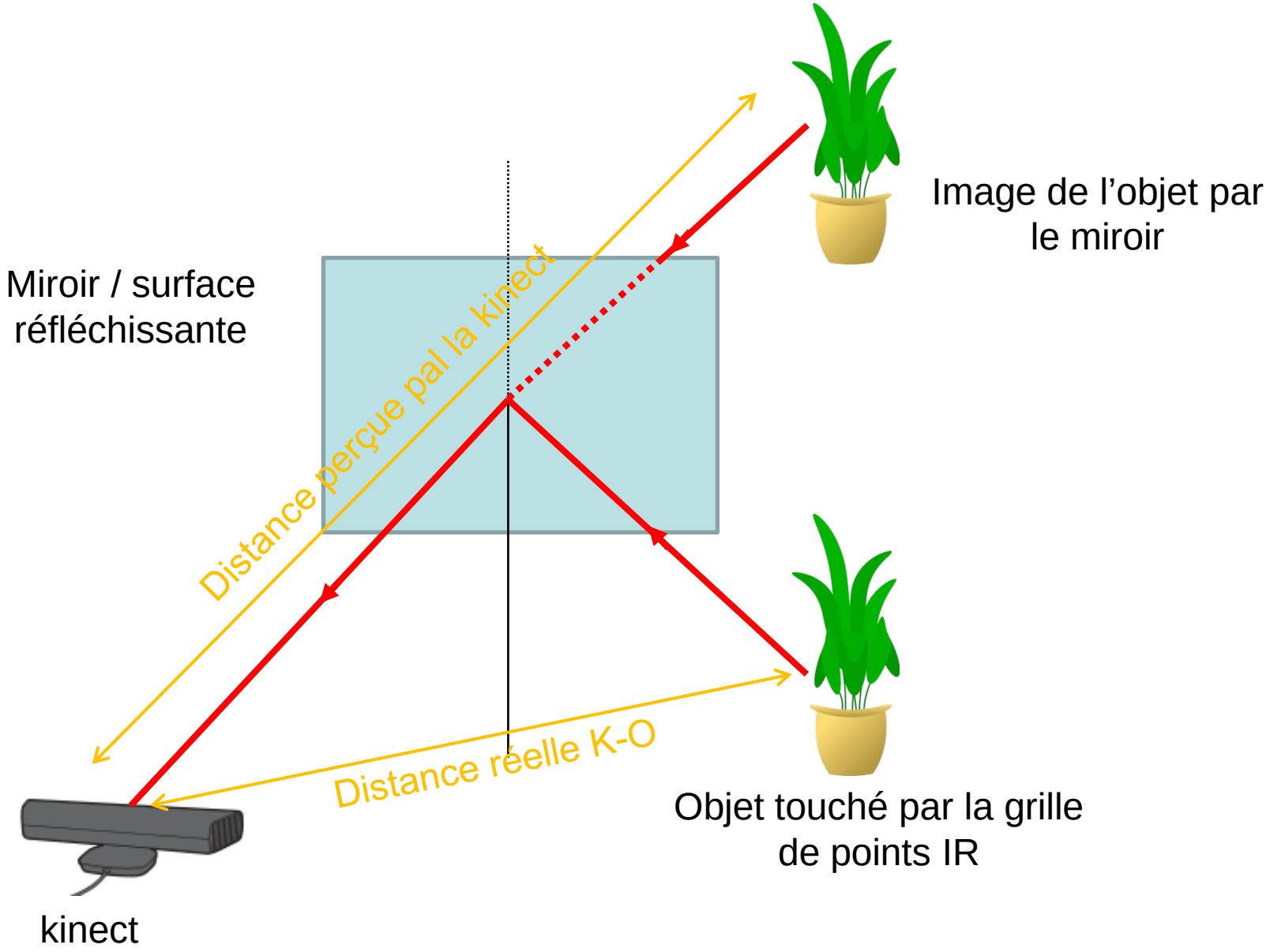
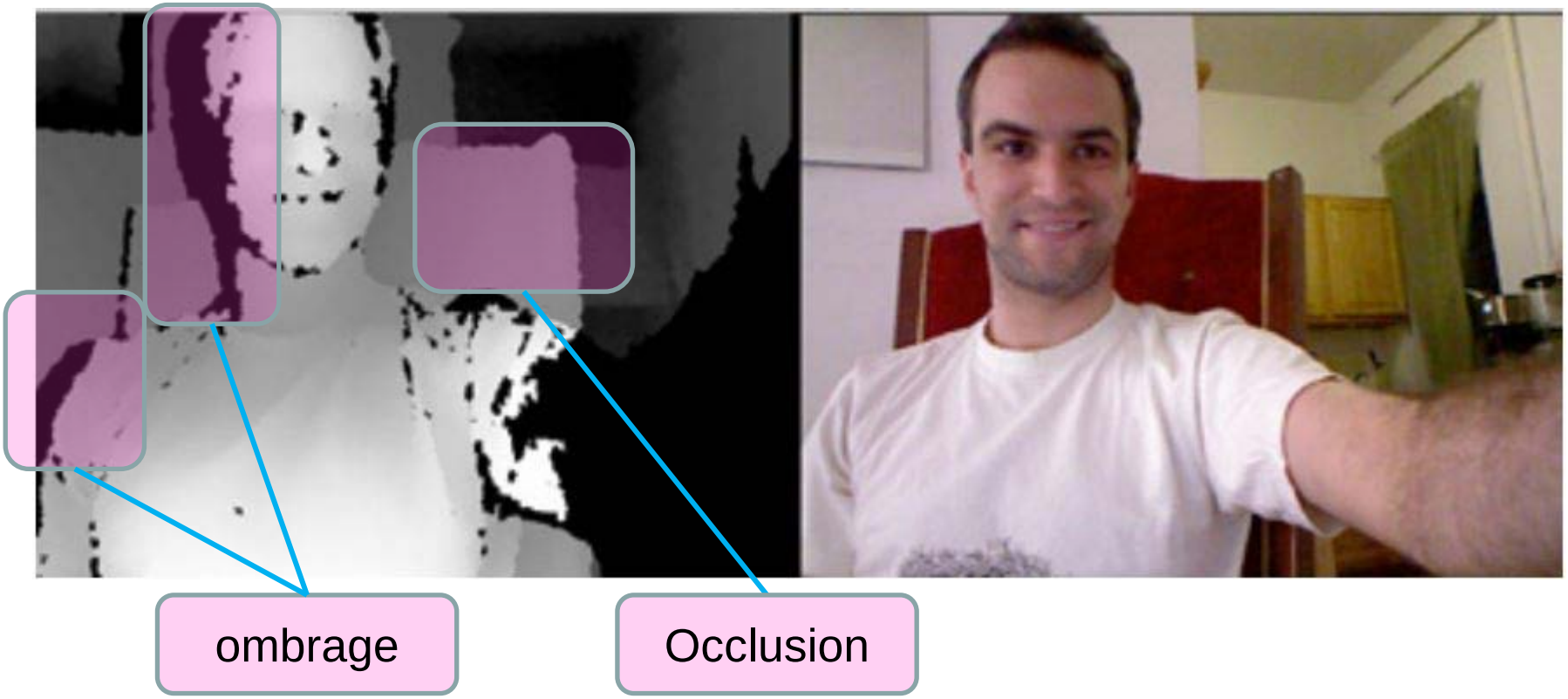


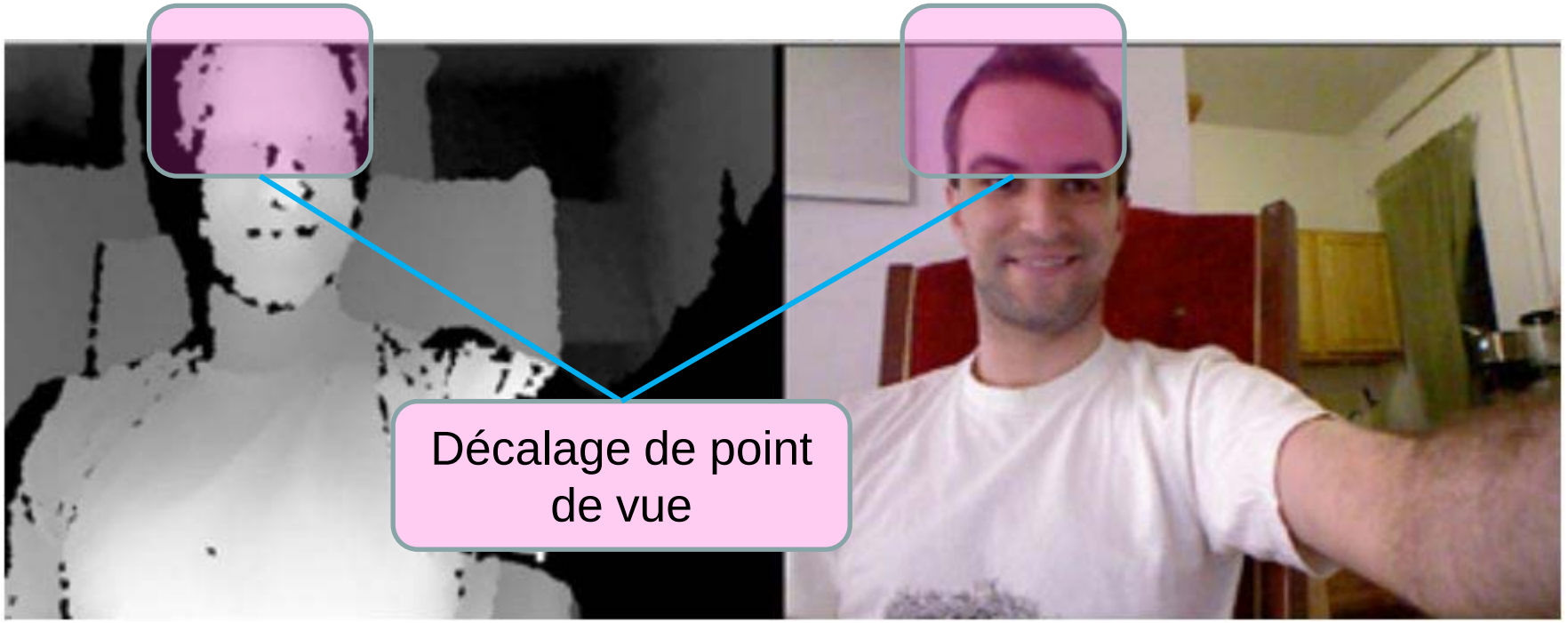
Image de l’objet par le miroir
Miroir / surface réfléchissante
Distance perçue pal la kinect
Distance réelle K-O
Objet touché par la grille de points IR
kinect

Occlusions et ombrages

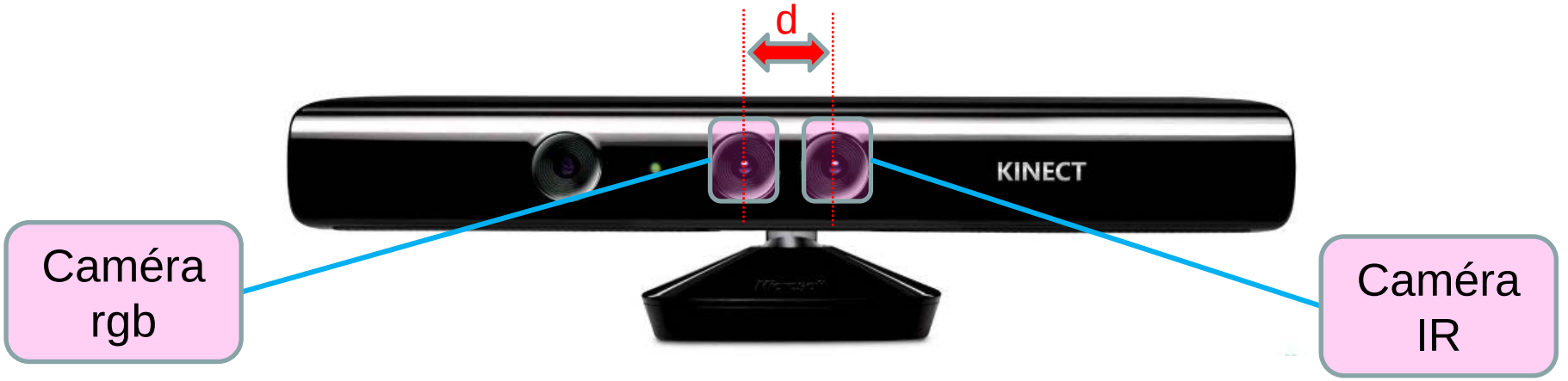


- Comme pour l'image en couleur, les zones qui ne sont pas éclairées par la grille IR ne peuvent être mesurées…
  - Zones ombragées
  - Zones d'occlusion : objets situés derrière des objets de 1er plan…

Désalignement entre l'image en couleur et l'image en profondeur



Décalage entre les images causé par le décalage entre la caméra rgb et la caméra IR:



2.3. Comprendre une image en profondeur

Image en profondeur
: image en NG
r = g = b

Image en couleur
(r,g,b)

Chaque canal sur 1 octet !!

Vérification par la lecture des pixels…

Programme suivant…

Lecture des pixels: (copier-coller)

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

void setup(){
size (640*2, 480);
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
kinect.enableRGB();
}

void draw(){
kinect.update();
PImage depthImage = kinect.depthImage();
PImage rgbImage = kinect.rgbImage();
image(depthImage, 0, 0);
image(rgbImage, 640, 0);
}

void mousePressed(){
color c = get(mouseX, mouseY);
println("r: " + red(c) + "g: " + green(c) + " b: " + blue(c));
}
```

Lecture des pixels: (copier-coller)

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

void setup(){
size (640*2, 480);
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
kinect.enableRGB();
}

void draw(){
kinect.update();
PImage depthImage = kinect.depthImage();
PImage rgbImage = kinect.rgbImage();
image(depthImage, 0, 0);
image(rgbImage, 640, 0);
}

void mousePressed(){
color c = get(mouseX, mouseY);
println("r: " + red(c) + "g: " + green(c) + " b: " + blue(c));
}
```

Vérifier la valeur des pixels de l'image en couleur et de l'image en profondeur

Q: comment relier la valeur d'un pixel à la distance réelle à un objet ?

- En réalité, l'image en profondeur est la représentation (très approximative) d'un **tableau de mesures effectuée par la kinect**:
  - Distances mesurées entre 0 (ou plutôt 50cm) et 8000mm…
  - Valeur de la mesure codée sur **11 bits** soit **2024 « paliers de distance »**
    - Pas de la mesure: $\Delta x = \frac{D_{max}}{2024} = \frac{8000}{2024} \approx 3,95\ mm$
- Les pixels de l'image en profondeur sont codés sur 1 octet
  - 256 « paliers de distance » traduits graphiquement en niveaux de gris
    - Pas de la mesure: $\Delta x' = \frac{8000}{256} \approx 31,3mm$ Résolution moindre !!

Possibilité d'accéder directement au tableau de mesure

Partie suivante ….

2.4. Accéder au tableau des distances

Bilan sur la classe SimpleOpenNI:

Classe
« **SimpleOpenNI** »

Méthodes déjà utilisées

- enableDepth()
- enableRGB()
- update()
- depthImage()
- rgbImage()

Attributs

Aucun pour l'instant…

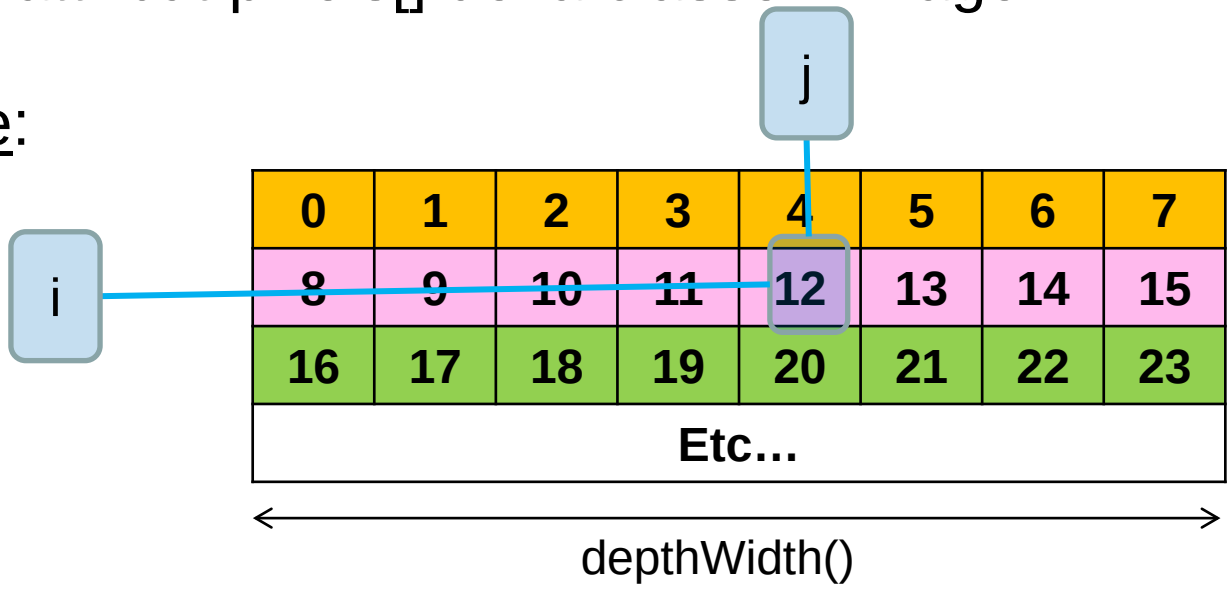
Nouvelle méthode

- depthMap()
  - Renvoie un tableau-1D (liste) d'entier contenant les différentes valeurs des profondeurs mesurées selon la grille IR.

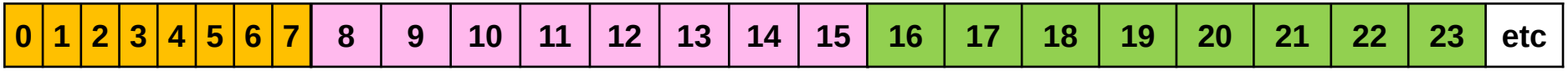
Organisation du tableau de profondeur renvoyé par depthMap()

⇨ Idem que l'attribut pixels[] de la classe PImage

depth Image:



depthMap :



⇨ Comment relier les coordonnées (i,j) d'un pixel à l'indice k du tableau depthMap ?

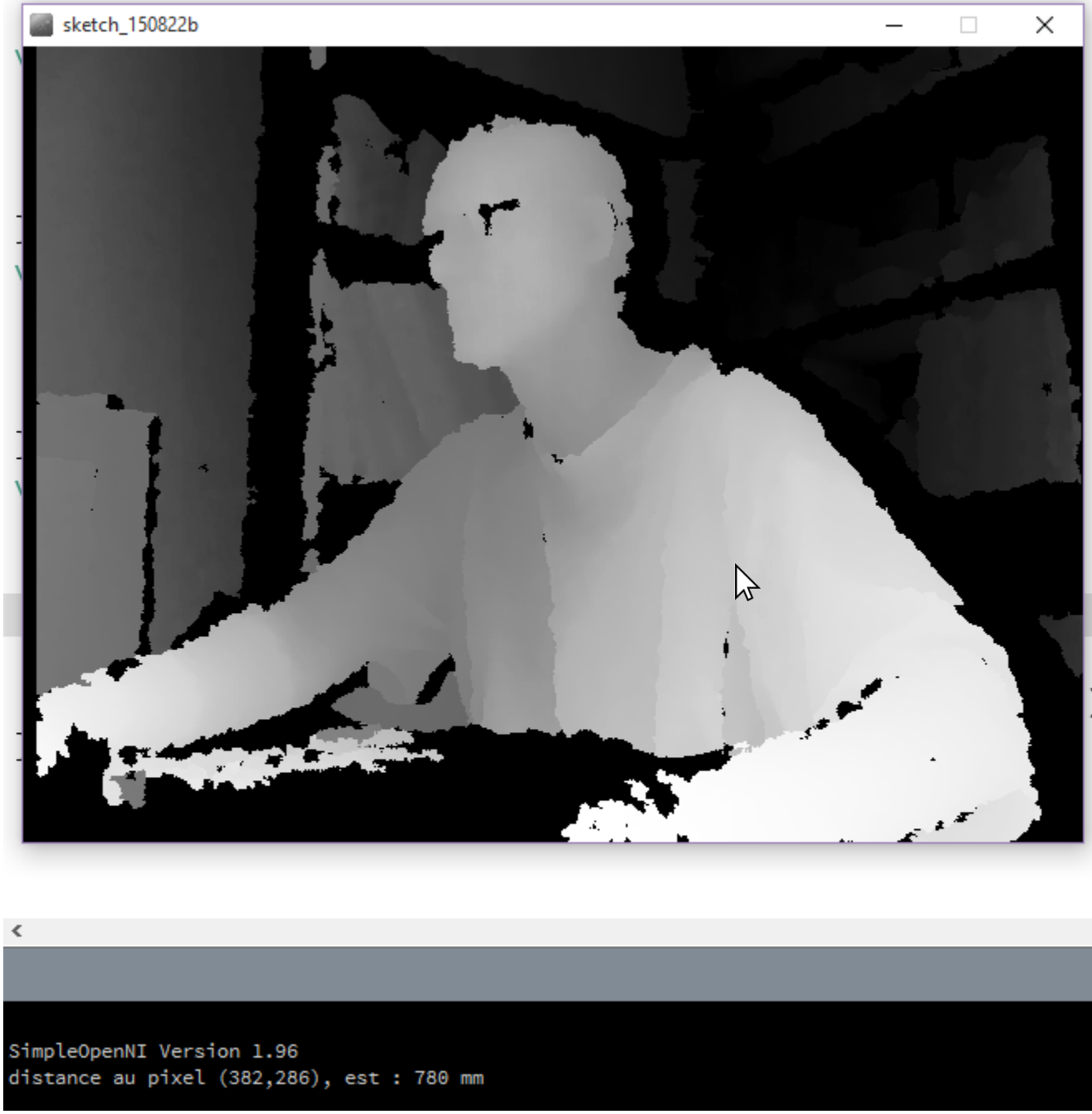
⇨ On montre : k = j + i * depthWidth()

Affichage de la distance d’un point à la caméra :

Algorithme:

- Afficher depthImage

- Lorsqu’on clique sur un pixel, écrire la valeur de la distance



Programme:

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

void setup() {
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw() {
  kinect.update();
  PImage depthImage = kinect.depthImage();
  image(depthImage, 0, 0);
}
void mousePressed(){
  int[] depthValues = kinect.depthMap();
  int clickPosition = mouseX + (mouseY * 640);
  int clickedDepth = depthValues[clickPosition];
  println("distance au pixel (" + mouseX + "," +
          mouseY + "), est : " + clickedDepth + " mm" );
}
```

Copier/coller:

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

void setup() {
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw() {
  kinect.update();
  PImage depthImage = kinect.depthImage();
  image(depthImage, 0, 0);
}
void mousePressed(){
  int[] depthValues = kinect.depthMap();
  int clickPosition = mouseX + (mouseY * 640);
  int clickedDepth = depthValues[clickPosition];
  println("distance au pixel (" + mouseX + "," +
          mouseY + "), est : " + clickedDepth + " mm" );
}
```

2.5. Traquer le point le plus proche de la kinect

Bilan sur la classe SimpleOpenNI:

Classe « **SimpleOpenNI** »

Méthodes déjà utilisées

- enableDepth()
- enableRGB()
- update()
- depthImage()
- rgbImage()

Attributs

Aucun pour l'instant…

Nouvelle méthode

- depthMap()

Renvoie un tableau-1D (liste) d'entier contenant les différentes valeurs des profondeurs mesurées selon la grille IR.

Algorithme de tracking pour chaque boucle « draw() :

- Récupérer le tableau des profondeurs depuis la kinect : depthMap()
- Parcourir les pixels de la depthImage – pour chaque ligne (indice y) :
  - pour chaque pixels de cette ligne (indice x) :
    - Calculer l'indice i dans le tableau depthMap (i=x+y*640)
    - Récupérer la valeur de la profondeur à cet indice i
      - Si cette valeur est la plus petite rencontrée:
        - Mémoriser cette plus petite valeur
        - Mémoriser la position la position correspondante (x,y) du pixel
  - Fin des deux boucles: À la fin du parcours de l'image, la dernière profondeur mémorisée est la plus petite de toute l'image et le dernier couple (x,y) mémorisé est la position correspondante
- Afficher l'image en profondeur
- Afficher un disque rouge à la position (x,y) correspondant à la plus petite profondeur précédemment détectée.

Programme

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
int closestValue;
int closestX;
int closestY;
void setup(){
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw(){
  closestValue = 8000;
  kinect.update();
  // get the depth array from the kinect
  int[] depthValues = kinect.depthMap();
  // for each row in the depth image
  for(int y = 0; y < 480; y++){
  // look at each pixel in the row
    for(int x = 0; x < 640; x++){
      // pull out the corresponding value from the depth array
      int i = x + y * 640;
      int currentDepthValue = depthValues[i];
      // if that pixel is the closest one we've seen so far
      if(currentDepthValue > 0 && currentDepthValue < closestValue){
      // save its value
        closestValue = currentDepthValue;
        // and save its position (both X and Y coordinates)
        closestX = x;
        closestY = y;
      }
    }
  }
  //draw the depth image on the screen
  image(kinect.depthImage(),0,0);
  // draw a red circle over it,
  // positioned at the X and Y coordinates
  // we saved of the closest pixel.
  fill(255,0,0);
  ellipse(closestX, closestY, 25, 25);
}
```

Copier/coller le code et lancer le programme

Précaution d'emploi:

Placer la kinect sur une table en hauteur et vérifier que dans la scène observée, l'objet le plus proche est bien l'objet souhaité (main, etc…)

Copier/coler:

```
import SimpleOpenNI.*;

SimpleOpenNI kinect;
int closestValue;
int closestX;
int closestY;

void setup(){
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
```

```
void draw(){
  closestValue = 8000;
  kinect.update();
  // get the depth array from the kinect
  int[] depthValues = kinect.depthMap();
  // for each row in the depth image
  for(int y = 0; y < 480; y++){
  // look at each pixel in the row
    for(int x = 0; x < 640; x++){
      // pull out the corresponding value from the depth array
      int i = x + y * 640;
      int currentDepthValue = depthValues[i];
      // if that pixel is the closest one we've seen so far
      if(currentDepthValue > 0 && currentDepthValue < closestValue){
      // save its value
        closestValue = currentDepthValue;
        // and save its position (both X and Y coordinates)
        closestX = x;
        closestY = y;
      }
    }
  }
  //draw the depth image on the screen
  image(kinect.depthImage(),0,0);
  // draw a red circle over it,
  // positioned at the X and Y coordinates
  // we saved of the closest pixel.
  fill(255,0,0);
  ellipse(closestX, closestY, 25, 25);
}
```

Commentaire du code

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
int closestValue;
int closestX;
int closestY;
void setup(){
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw(){
  closestValue = 8000;
  kinect.update();
  // get the depth array from the kinect
  int[] depthValues = kinect.depthMap();
  // for each row in the depth image
  for(int y = 0; y < 480; y++){
  // look at each pixel in the row
    for(int x = 0; x < 640; x++){
      // pull out the corresponding value from the depth array
      int i = x + y * 640;
      int currentDepthValue = depthValues[i];
      // if that pixel is the closest one we've seen so far
      if(currentDepthValue > 0 && currentDepthValue < closestValue){
      // save its value
        closestValue = currentDepthValue;
        // and save its position (both X and Y coordinates)
        closestX = x;
        closestY = y;
      }
    }
  }
  //draw the depth image on the screen
  image(kinect.depthImage(),0,0);
  // draw a red circle over it,
  // positioned at the X and Y coordinates
  // we saved of the closest pixel.
  fill(255,0,0);
  ellipse(closestX, closestY, 25, 25);
}
```

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
int closestValue;
int closestX;
int closestY;
void setup(){
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw(){
  closestValue = 8000;
  kinect.update();
  // get the depth array from the kinect
  int[] depthValues = kinect.depthMap();
  // for each row in the depth image
  for(int y = 0; y < 480; y++){
  // look at each pixel in the row
    for(int x = 0; x < 640; x++){
      // pull out the corresponding value from the depth array
      int i = x + y * 640;
      int currentDepthValue = depthValues[i];
      // if that pixel is the closest one we've seen so far
      if(currentDepthValue > 0 && currentDepthValue < closestValue){
      // save its value
        closestValue = currentDepthValue;
        // and save its position (both X and Y coordinates)
        closestX = x;
        closestY = y;
      }
    }
  }
  //draw the depth image on the screen
  image(kinect.depthImage(),0,0);
  // draw a red circle over it,
  // positioned at the X and Y coordinates
  // we saved of the closest pixel.
  fill(255,0,0);
  ellipse(closestX, closestY, 25, 25);
}
```

Variables globales
(par défaut initialisées à 0)

Initialisation/instanciation de la classe SimpleOpenNI et lancement de l'acquisition de l'image en profondeur

Initialisation de la plus petite valeur détectée à 8000 (la plus grande valeur possible)

Récupération des données issues de la kinect (depthImage)

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
int closestValue;
int closestX;
int closestY;
void setup(){
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw(){
  closestValue = 8000;
  kinect.update();
  // get the depth array from the kinect
  int[] depthValues = kinect.depthMap();
  // for each row in the depth image
  for(int y = 0; y < 480; y++){
  // look at each pixel in the row
    for(int x = 0; x < 640; x++){
      // pull out the corresponding value from the depth array
      int i = x + y * 640;
      int currentDepthValue = depthValues[i];
      // if that pixel is the closest one we've seen so far
      if(currentDepthValue > 0 && currentDepthValue < closestValue){
      // save its value
        closestValue = currentDepthValue;
        // and save its position (both X and Y coordinates)
        closestX = x;
        closestY = y;
      }
    }
  }
  //draw the depth image on the screen
  image(kinect.depthImage(),0,0);
  // draw a red circle over it,
  // positioned at the X and Y coordinates
  // we saved of the closest pixel.
  fill(255,0,0);
  ellipse(closestX, closestY, 25, 25);
}
```

Mémorisation du tableau de profondeur dans la variable depthValues[]

Recherche et mémorisation de la plus petite valeur de la profondeur parmi tous les pixels de depthImage

Affichage de depthImage

Affichage d'un disque rouge à la position de la plus petite profondeur

Amélioration possible du programme

- Utilisation d'une variable unique de type Pvector pour mémoriser la position du point le plus proche :

```
int closestX
int closestY
```
→ Pvector closestPosition
  → À initialiser dans le setup()

- Utilisation de la méthode *min()* directement sur le tableau depthValues[] pour déterminer la profondeur minimale…
  - On évite la double boucle
  - Pb: trouver la position de l'élément trouvé….

- Autre possibilité: garder la double boucle mais effectuer la recherche de la profondeur minimale et de sa position dans une méthode à part : findMinValue()

2.6. Dessiner au « feutre invisible » !!

- Utilisation de la distance minimale précédemment détectée pour dessiner une courbe à la main…
- Programme quasiment identique au précédent :
  - Rajout de variables globales permettant la mémorisation des positions (x_{min},y_{min}) précédemment trouvées
  - Tracer de proche en proche une ligne entre le point (x_{min},y_{min}) précédemment trouvés et le nouveau point (x_{min},y_{min}) nouvellement trouvé…

code:

```
import SimpleOpenNI.*;

SimpleOpenNI kinect;
int closestValue;
int closestX;
int closestY;

// declare global variables for the
// previous x and y coordinates
int lastX, lastY ;

void setup(){
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
```

```
void draw(){
  closestValue = 8000;
  kinect.update();
  // get the depth array from the kinect
  int[] depthValues = kinect.depthMap();
  // for each row in the depth image
  for(int y = 0; y < 480; y++){
  // look at each pixel in the row
    for(int x = 0; x < 640; x++){
      // pull out the corresponding value from the depth array
      int i = x + y * 640;
      int currentDepthValue = depthValues[i];
      // if that pixel is the closest one we've seen so far
      if(currentDepthValue > 0 && currentDepthValue < closestValue){
      // save its value
        closestValue = currentDepthValue;
        // and save its position (both X and Y coordinates)
        closestX = x;
        closestY = y;
      }
    }
  }
  //draw the depth image on the screen
  image(kinect.depthImage(),0,0);
  // set the line drawing color to red
  stroke(255,0,0);
  // draw a line from the previous point to the new closest one
  line(lastX, lastY, closestX, closestY);
  // save the closest point as the new previous one
  lastX = closestX;
  lastY = closestY;
}
```

Différents problèmes:

- Les lignes successives sont effacées à cause du rafraichissement de l'image en profondeur ….

- Solution:
  - Ne plus afficher depthImage dans le draw()
  - Afficher sur fond noir…

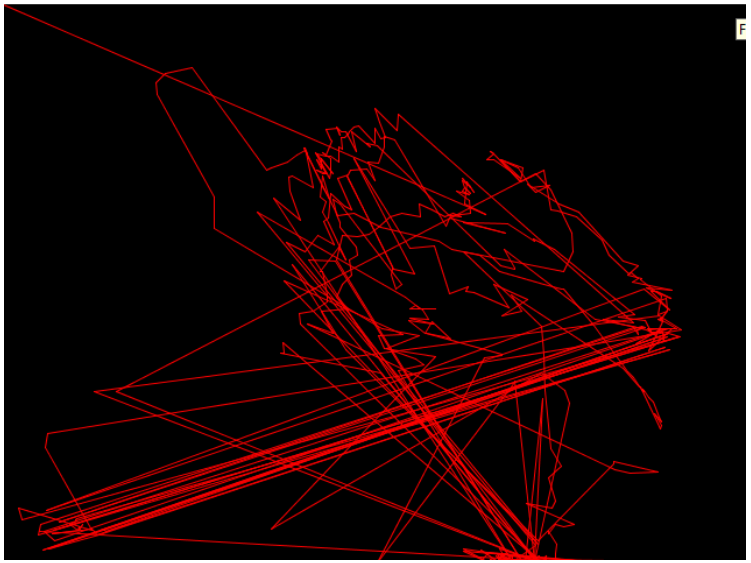
code:

```
import SimpleOpenNI.*;

SimpleOpenNI kinect;
int closestValue;
int closestX;
int closestY;

// declare global variables for the
// previous x and y coordinates
int lastX, lastY ;

void setup(){
  size(640, 480);
  background(0);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
```

```
void draw(){
  closestValue = 8000;
  kinect.update();
  // get the depth array from the kinect
  int[] depthValues = kinect.depthMap();
  // for each row in the depth image
  for(int y = 0; y < 480; y++){
  // look at each pixel in the row
    for(int x = 0; x < 640; x++){
      // pull out the corresponding value from the depth array
      int i = x + y * 640;
      int currentDepthValue = depthValues[i];
      // if that pixel is the closest one we've seen so far
      if(currentDepthValue > 0 && currentDepthValue < closestValue){
      // save its value
        closestValue = currentDepthValue;
        // and save its position (both X and Y coordinates)
        closestX = x;
        closestY = y;
      }
    }
  }
  //draw the depth image on the screen
 // image(kinect.depthImage(),0,0);
  // set the line drawing color to red
  stroke(255,0,0);
  // draw a line from the previous point to the new closest one
  line(lastX, lastY, closestX, closestY);
  // save the closest point as the new previous one
  lastX = closestX;
  lastY = closestY;
}
```

Différents problèmes:

- Sauts trop importants entre les différentes lignes !!



- Mauvais tracking

- Déplacement inversé par rapport à la réalité…
- Solution:
  - Symétriser la position x dans l'image : x ← 640 - x - 1
  - Ne prendre en compte que les positions entre 60cm et 1,5m pour éliminer les sauts trop grands
  - Interpoler les positions et lisser la courbe entre les différentes positions

code:

1

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
int closestValue;
int closestX;
int closestY;
float lastX;
float lastY;

void setup(){
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();

  // initialisation de la position initiale du point traqué
  // au milieu de la fenêtre d'affichage
  lastX = width/2;
  lastY = height/2;

  // start out with a black background
  background(0);
}
```

2

```
void draw(){
  closestValue = 8000;
  kinect.update();
  int[] depthValues = kinect.depthMap();
  for(int y = 0; y < 480; y++){
    for(int x = 0; x < 640; x++){
      // reverse x by moving in from
      // the right side of the image

      //int reversedX = 640-x-1;
      int reversedX = x;

      // use reversedX to calculate
      // the array index
      int i = reversedX + y * 640;
      int currentDepthValue = depthValues[i];
      // only look for the closestValue within a range
      // 610 (or 2 feet) is the minimum
      // 1525 (or 5 feet) is the maximum
      if(currentDepthValue > 610 && currentDepthValue < 1525
      && currentDepthValue < closestValue){
        closestValue = currentDepthValue;
        closestX = x;
        closestY = y;
      }
    }
  }
  // "linear interpolation", i.e.
  // smooth transition between last point
  // and new closest point
  float interpolatedX = lerp(lastX, closestX, 0.3f);
  float interpolatedY = lerp(lastY, closestY, 0.3f);
  stroke(255,0,0);

  // make a thicker line, which looks nicer
  strokeWeight(3);
  line(lastX, lastY, interpolatedX, interpolatedY);
  lastX = interpolatedX;
  lastY = interpolatedY;
}
```

3

```
void mousePressed(){
  // save image to a file
  // then clear it on the screen
  save("drawing.png");
  background(0);
}
```

2.7. Manipulation d’image par le geste

Projet 7: Minority Report Photos

⇨ Voir tutoriel ….

CHAPITRE 3

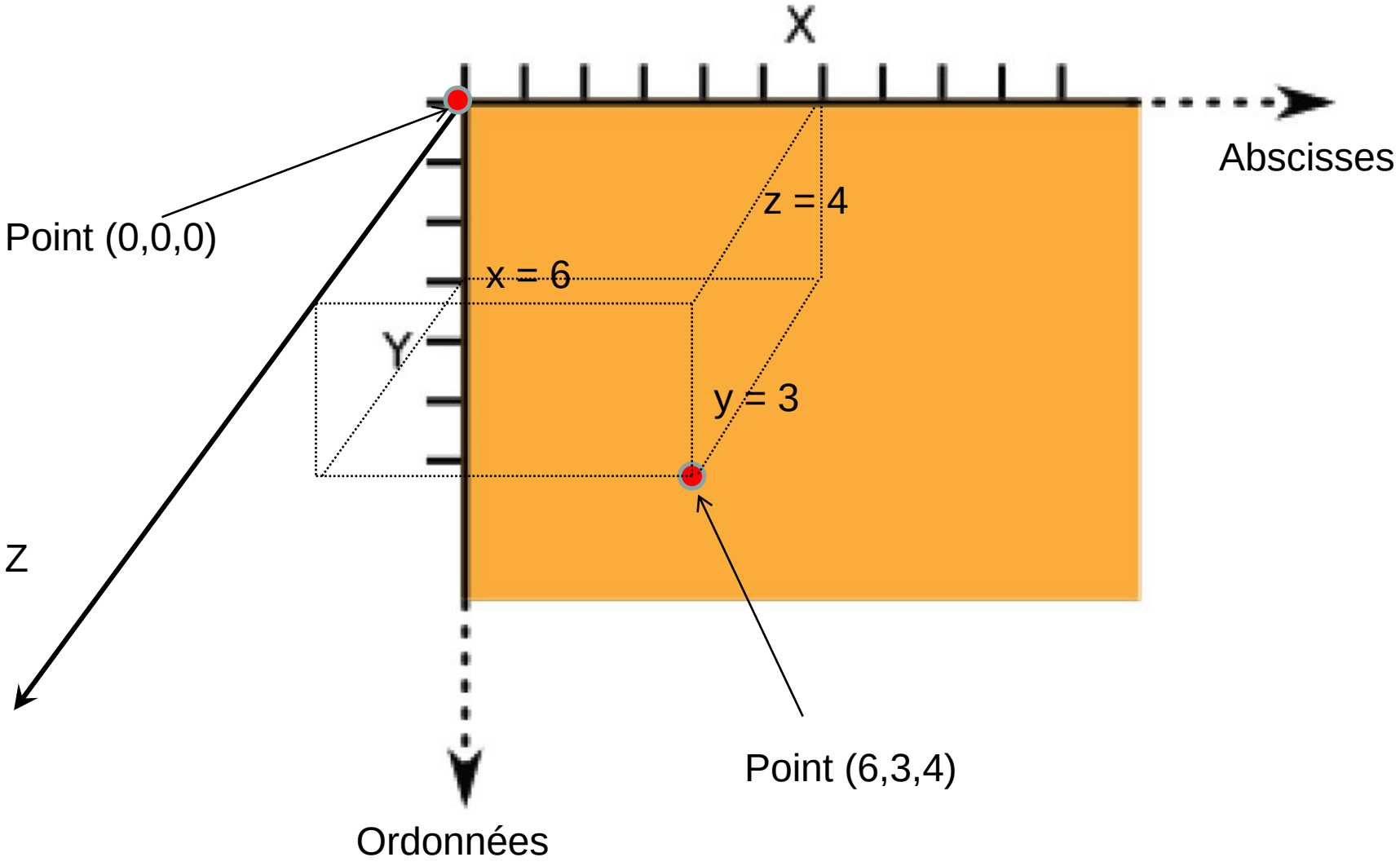
TRAVAILLER SUR LE NUAGE DE POINTS

III. TRAVAILLER SUR LE NUAGE DE POINTS

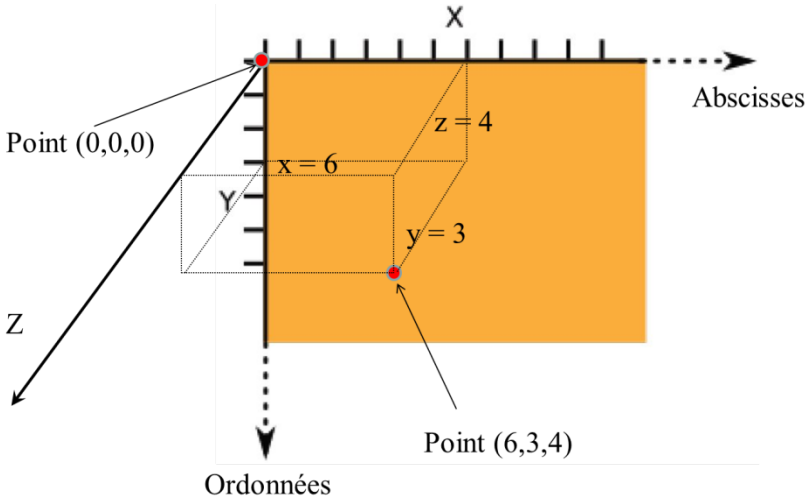
3.1. Présentation – espace 3D

La Kinect renvoie les coordonnées 3D des points détectés

Création d'un modèle 3D de la scène !!



- La Kinect renvoie les coordonnées 3D des points détectés
  - Création d'un modèle 3D de la scène !!



  - Possibilité de faire tourner la scène et de l'observer sous différents angles de vue, sans bouger la kinect !!!
- Voir le projet « KINECT/MULTITOUCH » effectué à la Gaité Lyrique (2015) par Morgane Guillaume et Audrey Herd-Smith
  - http://www.tonerkebab.fr/wiki/doku.php/wiki:projets:gaite:projet1

3.2. Affichage du nuage de points

Classe
« **SimpleOpenNI** »

Méthodes déjà utilisées

- enableDepth()
- enableRGB()
- update()
- depthImage()
- rgbImage()
- depthMap()

Attributs

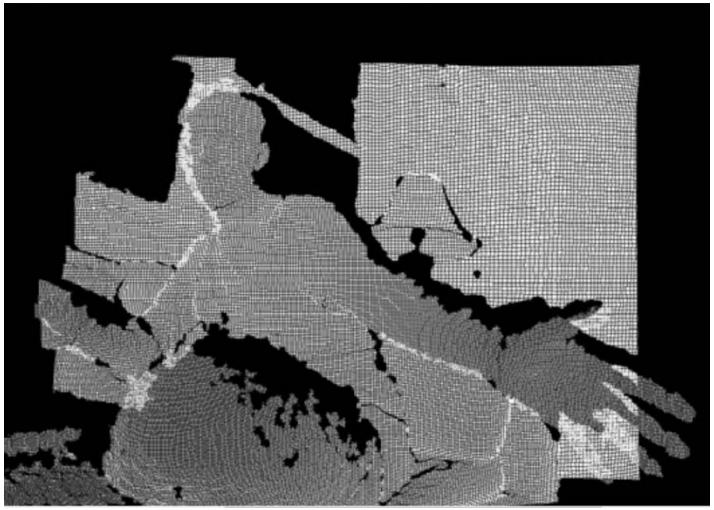
Aucun pour l'instant…

Nouvelle méthode

- depthMapRealWorld()
  - Renvoie un tableau-1D (liste) de PVector contenant les coordonnées 3D des points de la scène…

```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}

void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 1000 pixels closer on z
  translate(width/2, height/2, -1000);
  rotateX(radians(180)); // flip y-axis from "realWorld" 2
  stroke(255);
  // get the depth data as 3D points
  PVector[] depthPoints = kinect.depthMapRealWorld();
  for(int i = 0; i < depthPoints.length; i++){
    // get the current point from the point array
    PVector currentPoint = depthPoints[i];
    // draw the current point
    point(currentPoint.x, currentPoint.y, currentPoint.z);
   }
}
```

Copier/coller le code et lancer le programme



Utilisation de la classe PVector !!

Commentaire du code

```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}

void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 1000 pixels closer on z
  translate(width/2, height/2, -1000);
  rotateX(radians(180)); // flip y-axis from "realWorld" 2
  stroke(255);
  // get the depth data as 3D points
  PVector[] depthPoints = kinect.depthMapRealWorld();
  for(int i = 0; i < depthPoints.length; i++){
    // get the current point from the point array
    PVector currentPoint = depthPoints[i];
    // draw the current point
    point(currentPoint.x, currentPoint.y, currentPoint.z);
   }
}
```

Librairies:
OpenGL (calcul d'image 3D)
SimpleOpenNI (Kinect)

Déclaration d'une variable globale kinect de type SimpleOpenNI

Étalonnage de la taille de la fenêtre d'affichage et appel à la 3D via OPENGL

Commentaire du code

```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}

void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 1000 pixels closer on z
  translate(width/2, height/2, -1000);
  rotateX(radians(180)); // flip y-axis from "realWorld" 2
  stroke(255);
  // get the depth data as 3D points
  PVector[] depthPoints = kinect.depthMapRealWorld();
  for(int i = 0; i < depthPoints.length; i++){
    // get the current point from the point array
    PVector currentPoint = depthPoints[i];
    // draw the current point
    point(currentPoint.x, currentPoint.y, currentPoint.z);
   }
}
```

Initialisation de l'objet kinect par instanciation de la classe SimpleOpenNI

lancement de l'acquisition de l'image en profondeur pour la récupération des données 3D

Fond noir

Récupération des données 3D issues de la kinect (depthMapRealWorld)

Centrage de l'affichage au milieu de la fenêtre d'affichage et translaté de 1000 pixels vers l'arrière de l'écran

Commentaire du code

```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}

void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 1000 pixels closer on z
  translate(width/2, height/2, -1000);
  rotateX(radians(180)); // flip y-axis from "realWorld" 2
  stroke(255);
  // get the depth data as 3D points
  PVector[] depthPoints = kinect.depthMapRealWorld();
  for(int i = 0; i < depthPoints.length; i++){
    // get the current point from the point array
    PVector currentPoint = depthPoints[i];
    // draw the current point
    point(currentPoint.x, currentPoint.y, currentPoint.z);
   }
}
```

L'axe des y est orienté vers le bas sous Processing, et orienté vers le haut dans la réalité => nécessité d'effectuer une rotation de 180° de l'affichage pour obtenir une image droite…
(faire le test en mettant en commentaire cette ligne)

Contours blanc

Commentaire du code

```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}

void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 1000 pixels closer on z
  translate(width/2, height/2, -1000);
  rotateX(radians(180)); // flip y-axis from "realWorld" 2
  stroke(255);
  // get the depth data as 3D points
  PVector[] depthPoints = kinect.depthMapRealWorld();
  for(int i = 0; i < depthPoints.length; i++){
    // get the current point from the point array
    PVector currentPoint = depthPoints[i];
    // draw the current point
    point(currentPoint.x, currentPoint.y, currentPoint.z);
   }
}
```

Appel à la méthode depthMapRealWorld() de la classe SimpleOpenNI qui renvoie le tableau 1D de Pvectors contenant les coordonnées 3D des points de la scène. Le tableau est placé dans la variable locale depthPoints[]…

Commentaire du code

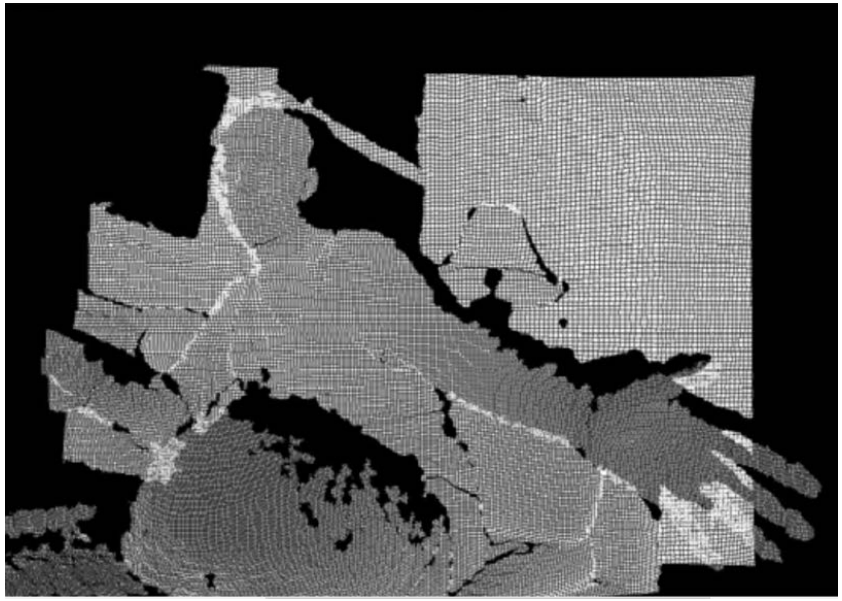
```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}

void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 1000 pixels closer on z
  translate(width/2, height/2, -1000);
  rotateX(radians(180)); // flip y-axis from "realWorld" 2
  stroke(255);
  // get the depth data as 3D points
  PVector[] depthPoints = kinect.depthMapRealWorld();
  for(int i = 0; i < depthPoints.length; i++){
    // get the current point from the point array
    PVector currentPoint = depthPoints[i];
    // draw the current point
    point(currentPoint.x, currentPoint.y, currentPoint.z);
   }
}
```

Affichage en 3D de tous les points contenus dans le tableau depthPoints

3.3. Mettre en mouvement automatiquement le nuage de points

problème:

- L'échantillonnage spatial de la kinect est très fin => les données à traiter sont très importantes



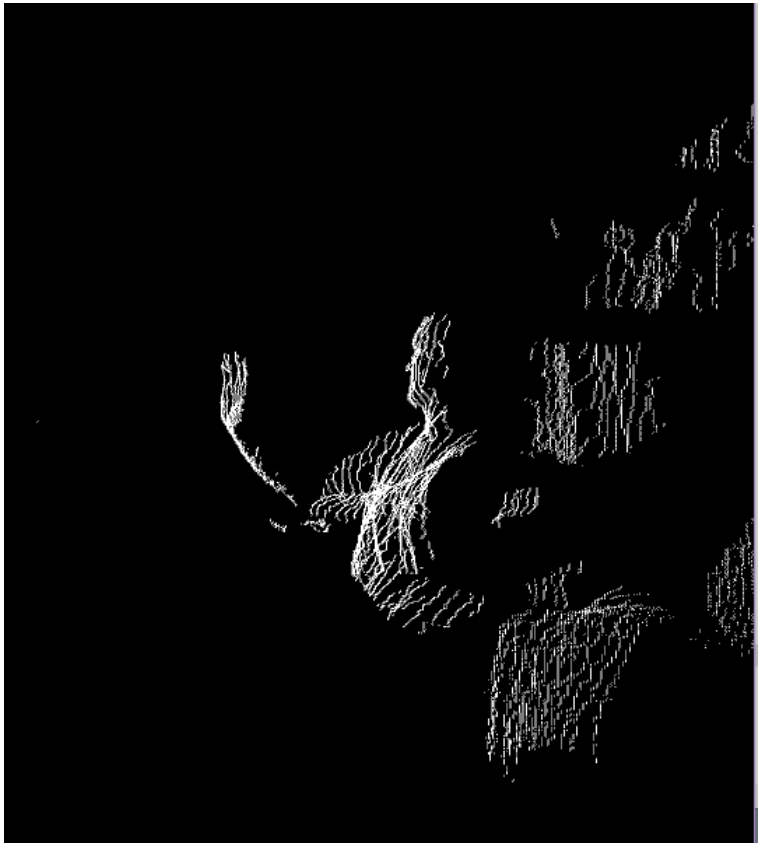
  - Risque de ralentir l'exécution en cas de mouvement…

- Solution:

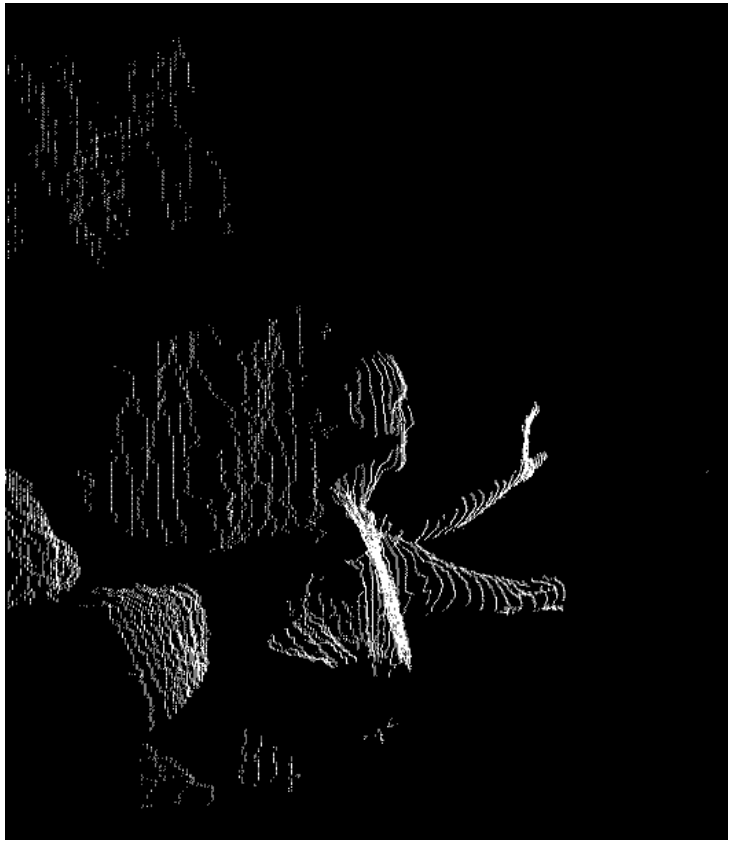
  - Tracer uniquement un 1 sur 10 du tableau renvoyé par kinect.depthMapRealWorld()

Objectif du programme

- Effectuer une rotation automatique de la scène par rapport à un axe vertical appartenant au plan de l'écran et passant par son centre :

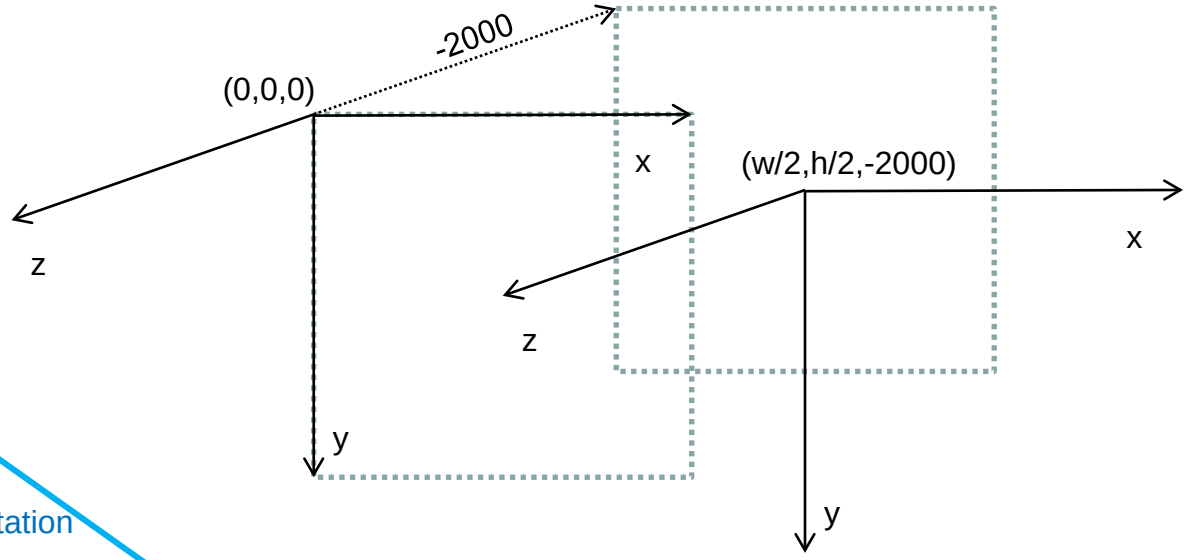


Rotation automatique



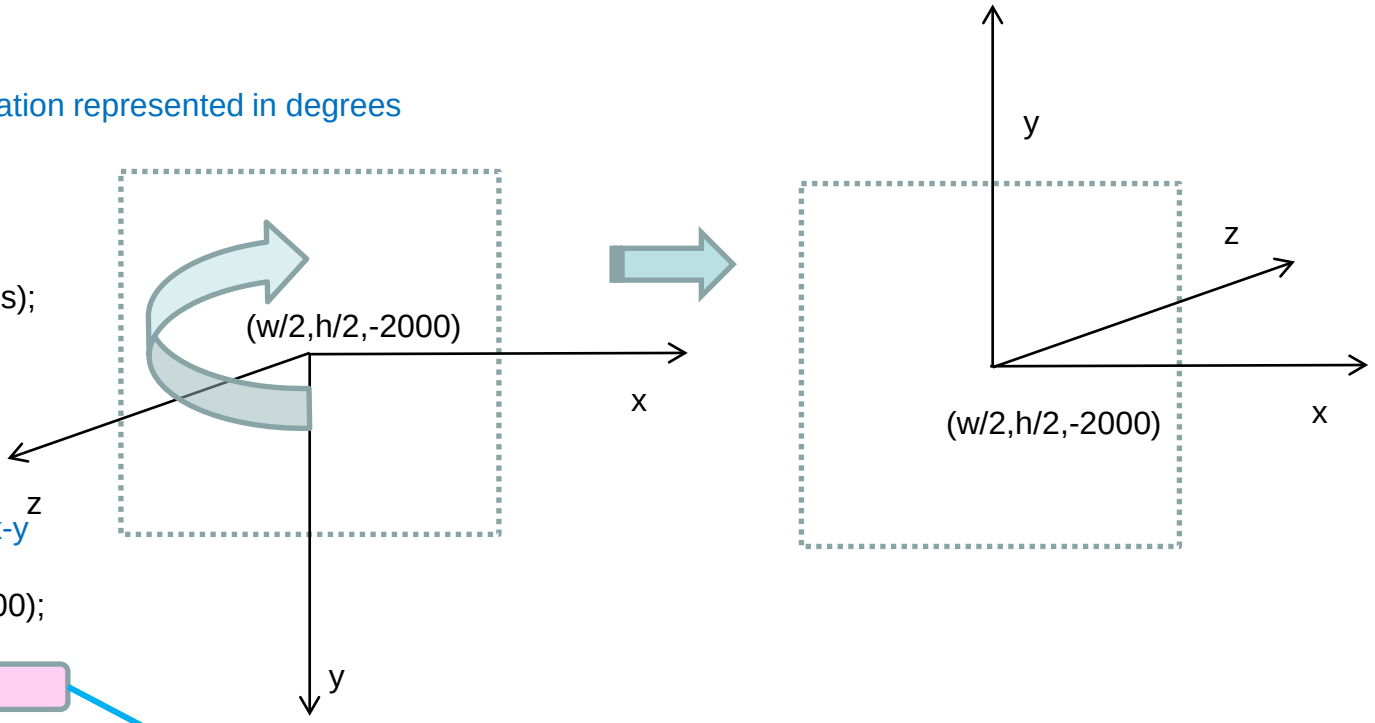
Copier/coller le code et lancer le programme

```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

// variable to hold our current rotation represented in degrees
float rotation = 0;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 2000 pixels closer on z
  translate(width/2, height/2, -2000);
  // flip the point cloud vertically:
  rotateX(radians(180));
// rotate about the y-axis and bump the rotation
  rotateY(radians(rotation));
  rotation++;
  stroke(255);
  PVector[] depthPoints = kinect.depthMapRealWorld();
  // notice: "i+=10"
  // only draw every 10th point to make things faster
  for (int i = 0; i < depthPoints.length; i+=10) {
    PVector currentPoint = depthPoints[i];
    point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
}
```



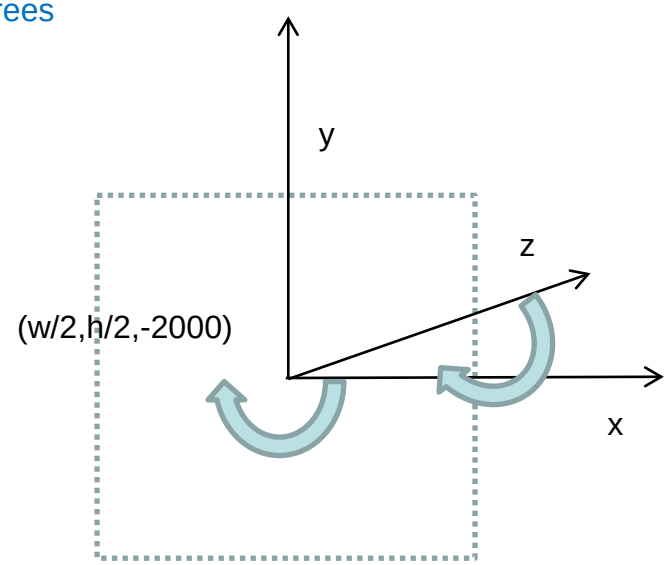
Translation de l'origine de (0,0,0) à (w/2,h/2,-2000) pour que l'on puisse observer tout le nuage de points lors de la rotation, sans sortir du champ

```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

// variable to hold our current rotation represented in degrees
float rotation = 0;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 2000 pixels closer on z
  translate(width/2, height/2, -2000);
  // flip the point cloud vertically:
  rotateX(radians(180));

// rotate about the y-axis and bump the rotation
  rotateY(radians(rotation));
  rotation++;
  stroke(255);
  PVector[] depthPoints = kinect.depthMapRealWorld();
  // notice: "i+=10"
  // only draw every 10th point to make things faster
  for (int i = 0; i < depthPoints.length; i+=10) {
    PVector currentPoint = depthPoints[i];
    point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
}
```



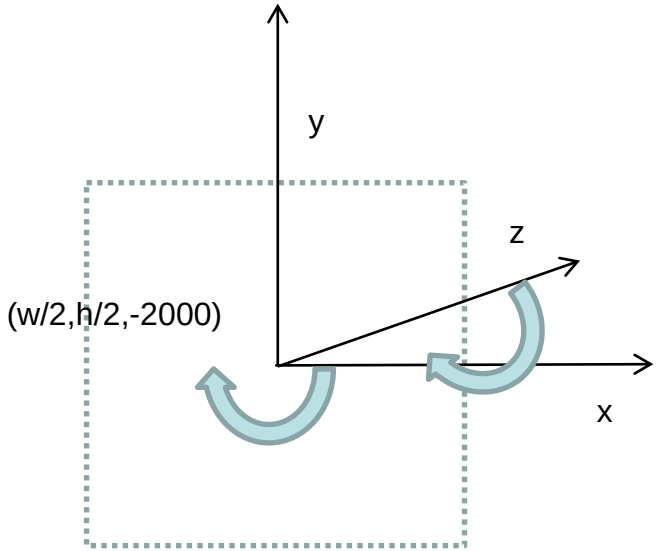
Rotation de 180° autour de l'axe des X

```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

// variable to hold our current rotation represented in degrees
float rotation = 0;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 2000 pixels closer on z
  translate(width/2, height/2, -2000);
  // flip the point cloud vertically:
  rotateX(radians(180));

// rotate about the y-axis and bump the rotation
  rotateY(radians(rotation));
  rotation++;
  stroke(255);
  PVector[] depthPoints = kinect.depthMapRealWorld();
  // notice: "i+=10"
  // only draw every 10th point to make things faster
  for (int i = 0; i < depthPoints.length; i+=10) {
    PVector currentPoint = depthPoints[i];
    point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
}
```



Rotation de d'un angle
« rotation » autour de l'axe y

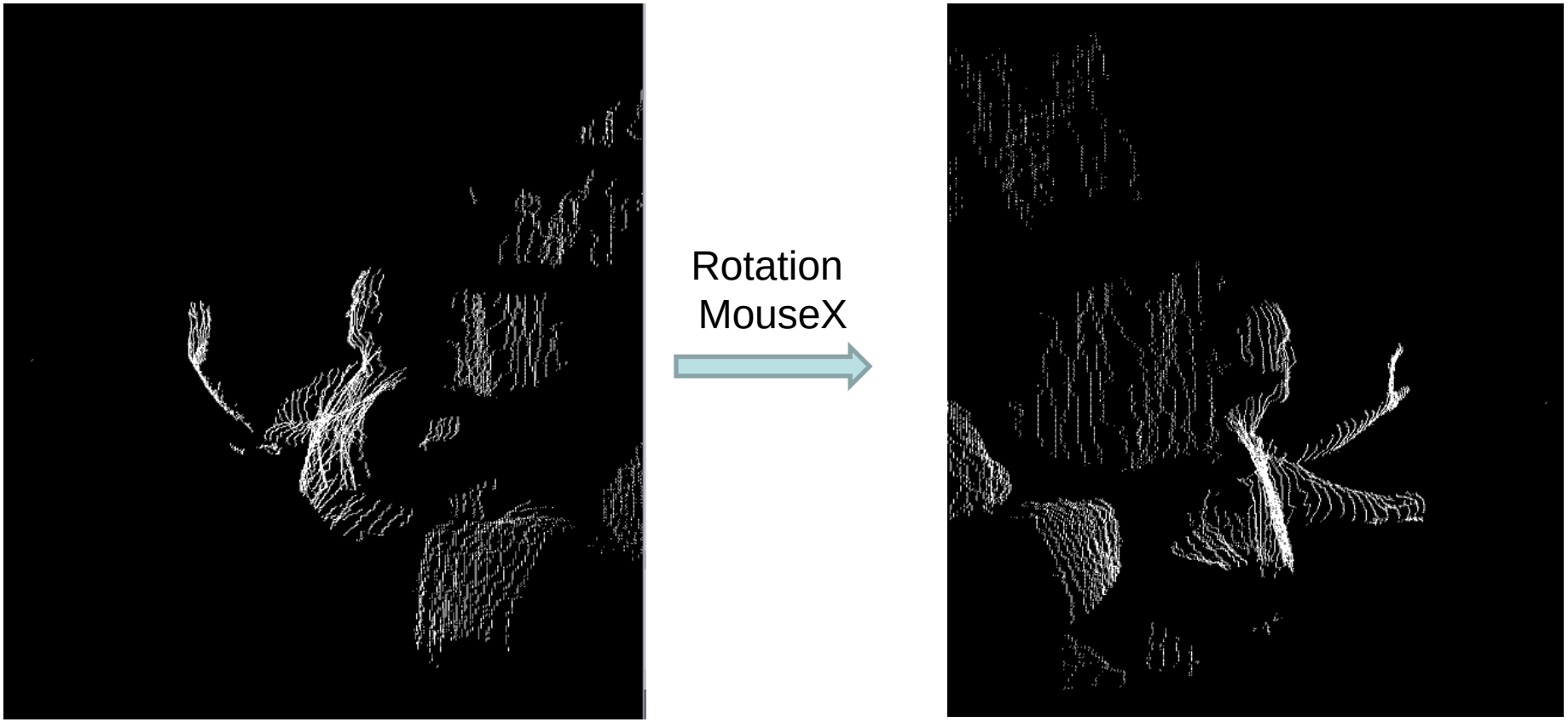
```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

// variable to hold our current rotation represented in degrees
float rotation = 0;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 2000 pixels closer on z
  translate(width/2, height/2, -2000);
  // flip the point cloud vertically:
  rotateX(radians(180));

// rotate about the y-axis and bump the rotation
  rotateY(radians(rotation));
  rotation++;
  stroke(255);
  PVector[] depthPoints = kinect.depthMapRealWorld();
  // notice: "i+=10"
  // only draw every 10th point to make things faster
  for (int i = 0; i < depthPoints.length; i+=10) {
    PVector currentPoint = depthPoints[i];
    point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
}
```



Affichage en 3D de tous les points contenus dans le tableau depthPoints

3.4. Mettre en mouvement le nuage de points à l'aide de la sourie

- Effectuer une rotation de la scène par rapport à un axe vertical appartenant au plan de l'écran et passant par son centre, A L'AIDE DE LA SOURIE !!



Copier/coller le code et lancer le programme

```
import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI kinect;

// variable to hold our current rotation represented in degrees
float rotation = 0;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw() {
  background(0);
  kinect.update();
  // prepare to draw centered in x-y
  // pull it 2000 pixels closer on z
  translate(width/2, height/2, -2000);
  // flip the point cloud vertically:
  rotateX(radians(180));

// rotate about the y-axis and bump the rotation with the mouse
rotation = map(mouseX,0,width,-180,180);
rotateY(radians(rotation));

stroke(255);
  PVector[] depthPoints = kinect.depthMapRealWorld();
  // notice: "i+=10"
  // only draw every 10th point to make things faster
  for (int i = 0; i < depthPoints.length; i+=10) {
    PVector currentPoint = depthPoints[i];
    point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
}
```

Modifier les deux lignes avant « stroke(255) »

Programme quasi-identique au précédent

3.5. Colorer le nuage de points

Objectif:

⇨ Colorer chaque point du nuage de points avec la couleur du pixel correspondant dans l'image RGB…

Copier/coller le code et lancer le programme

```
import processing.opengl.*;
import SimpleOpenNI.*;
SimpleOpenNI kinect;
float rotation = 0;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
  // access the color camera
  kinect.enableRGB();
  // tell OpenNI to line-up the color pixels with the depth data
  kinect.alternativeViewPointDepthToImage();
}
void draw() {
  background(0);
  kinect.update();
  // load the color image from the Kinect
  PImage rgbImage = kinect.rgbImage();
  //translate(width/2, height/2, -250);
  translate(width/2, height/2, -1250);
  rotateX(radians(180));
  //translate(0, 0, 1000);

  // régage de l'angle de vue avec la sourie
  rotation = map(mouseX,0,width,-180,180);
  rotateY(radians(rotation));

  // dessin des points en 3D
  PVector[] depthPoints = kinect.depthMapRealWorld();
  // don't skip any depth points
  for (int i = 0; i < depthPoints.length; i+=3) {
    PVector currentPoint = depthPoints[i];
    // set the stroke color based on the color pixel
    stroke(rgbImage.pixels[i]);
    point(currentPoint.x, currentPoint.y, currentPoint.z);  } }
```

Mise en route de la camera couleur

appel à la méthode alternativeViewPointDepthToImage() pour aligner les pixels en couleurs sur ceux de l'image en profondeur

Étalonage de la couleur du point dessiné avec la couleur du pixel (RGB) correspondant

3.6. Nuage de points interactif

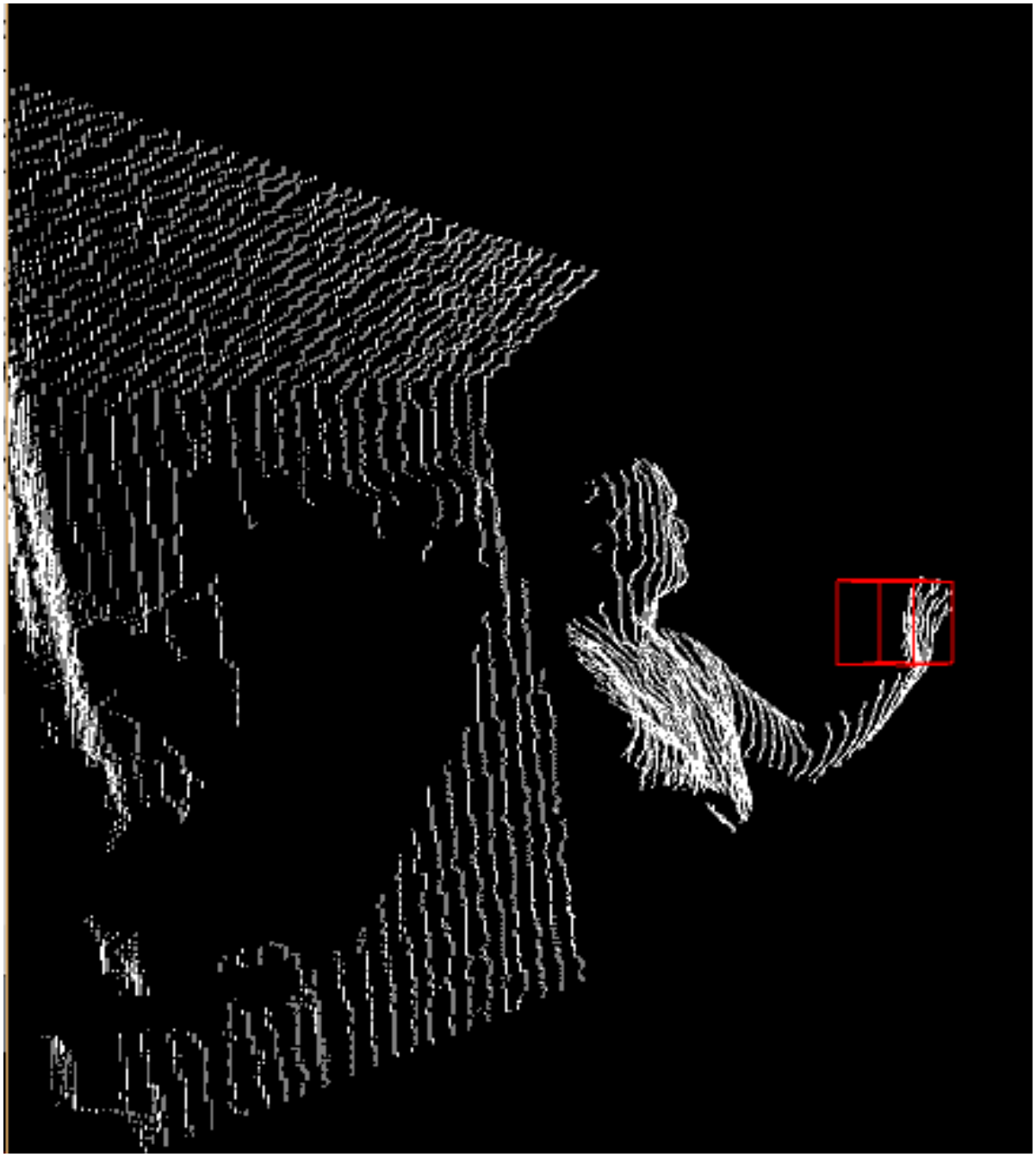
Objectif:

- Modéliser un espace 3D interactif (ex: cube, sphère)
- Tester si des points du nuage appartiennent à ce volume
- Lancer une action si un/plusieurs points appartiennent au volume (ex: musique, couleurs, etc.)

↳ Projet ultime et complexe: « KINECT/MULTITOUCH » effectué à la Gaité Lyrique (2015) par Morgane Guillaume et Audrey Herd-Smith

http://www.tonerkebab.fr/wiki/doku.php/wiki:projets:gaite:projet1

1) Tracé du cube



code

```
import processing.opengl.*;
import SimpleOpenNI.*;
SimpleOpenNI kinect;
float rotation = 0;
// set the box size
int boxSize = 150;
// a vector holding the center of the box
PVector boxCenter = new PVector(0, 0, 600);

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}
void draw() {
  background(0);
  kinect.update();
  translate(width/2, height/2, -2000);
  rotateX(radians(180));
  rotateY(radians(map(mouseX, 0, width, -180, 180)));
  stroke(255);
  PVector[] depthPoints = kinect.depthMapRealWorld();
  for (int i = 0; i < depthPoints.length; i+=10) {
  PVector currentPoint = depthPoints[i];
  point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
  // We're ready to draw the cube
  // move to the box center
  translate(boxCenter.x, boxCenter.y, boxCenter.z);
  // set line color to red
  stroke(255, 0, 0);
  // leave the box unfilled so we can see through it
  noFill();
  // draw the box
  box(boxSize);
}
```

Taille de l'arête du cube

Coordonnées du centre du cube du cube

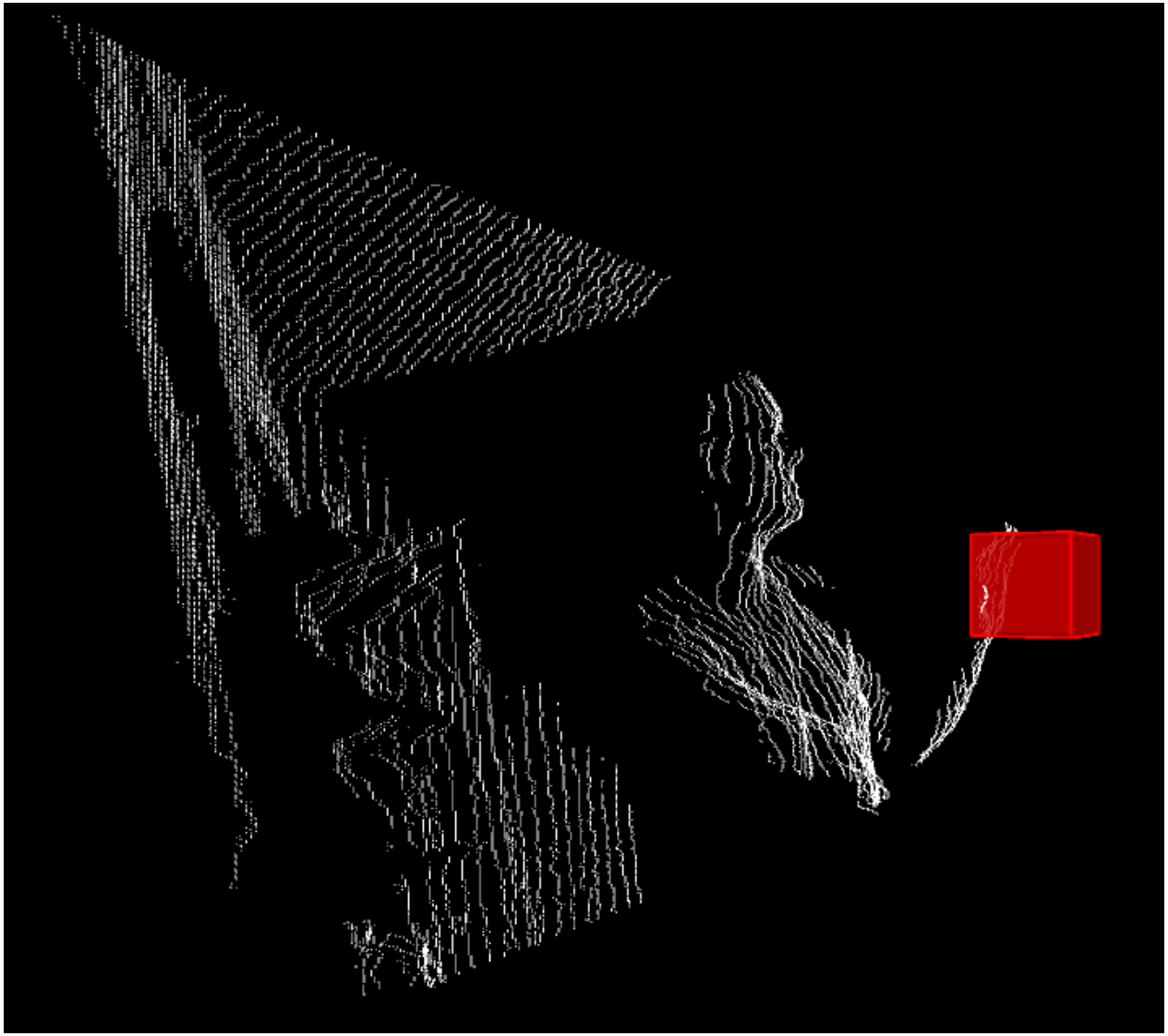
Code identique au sketchs précédants

Positionnement du centre du cube à la position souhaitée avant dessin

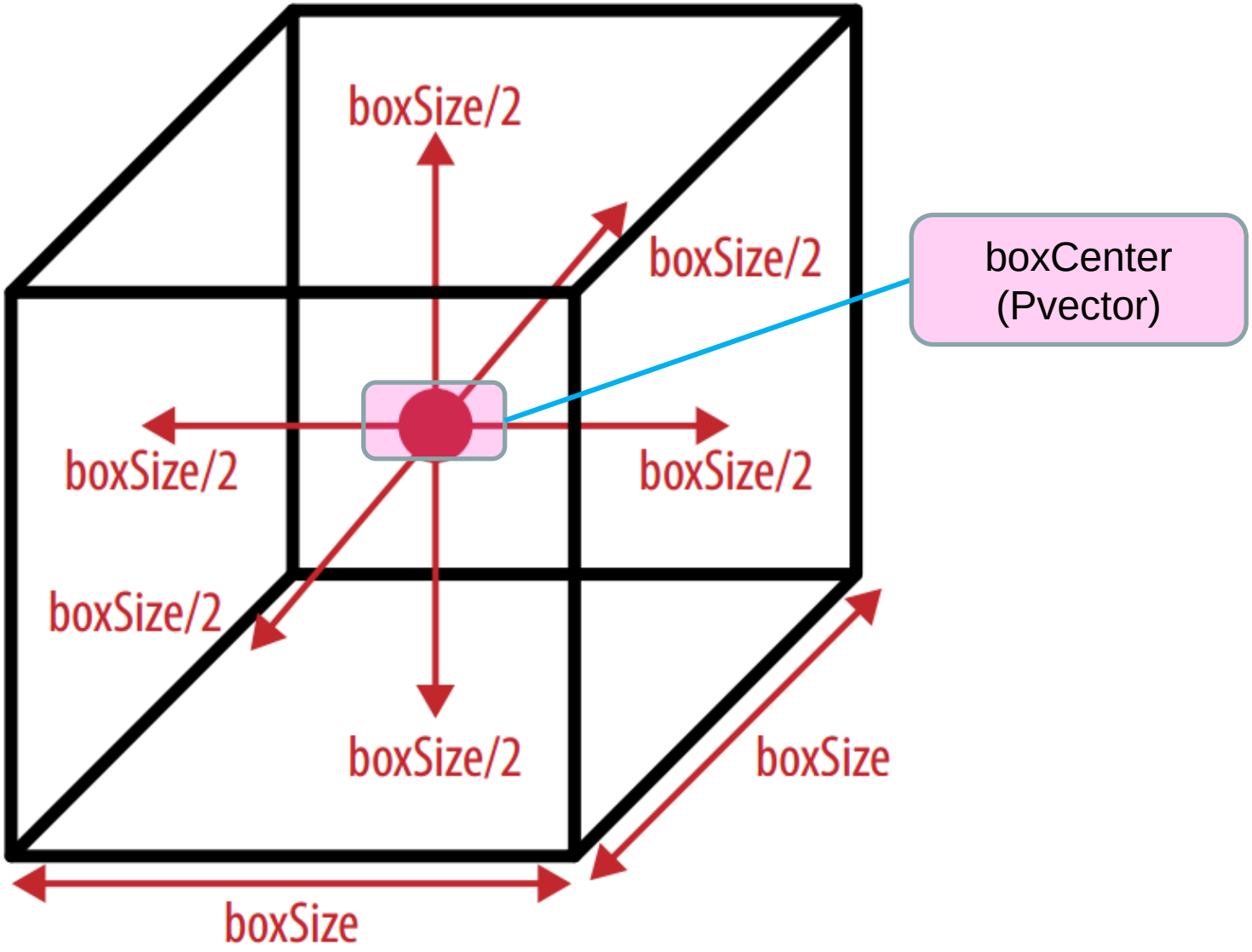
Dessin du cube en rouge non rempli…

2) Test du nombre de points contenus dans le cube

Objectif : Plus le nombre de points du nuage contenu dans le cube est important, plus l'opacité augmente….



Algorithme du test du nombre de points contenus dans le cube :



code

```
import processing.opengl.*;
import SimpleOpenNI.*;
SimpleOpenNI kinect;
float rotation = 0;
int boxSize = 150;
PVector boxCenter = new PVector(0, 0, 600);
// this will be used for zooming
// start at normal
float s = 1;
void setup() {
size(1024, 768, OPENGL);
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
}
```

2

```
void draw() {
  background(0);
  kinect.update();
  translate(width/2, height/2, -1000);
  rotateX(radians(180));
  // bumped up the translation
  // so that scale is better centered
  translate(0, 0, 1400);
  rotateY(radians(map(mouseX, 0, width, -180, 180)));
  // make everything bigger, i.e. zoom in
  translate(0,0,s*-1000);
  scale(s);
  println(s);
  stroke(255);
  PVector[] depthPoints = kinect.depthMapRealWorld();
  // initialize a variable
  // for storing the total
  // points we find inside the box
  // on this frame
  int depthPointsInBox = 0;
  for (int i = 0; i < depthPoints.length; i+=10) {
    PVector currentPoint = depthPoints[i];
    // The nested if statements inside of our loop
    if (currentPoint.x > boxCenter.x - boxSize/2
    && currentPoint.x < boxCenter.x + boxSize/2)
    {
      if (currentPoint.y > boxCenter.y - boxSize/2
      && currentPoint.y < boxCenter.y + boxSize/2)
      {
        if (currentPoint.z > boxCenter.z - boxSize/2
        && currentPoint.z < boxCenter.z + boxSize/2)
        {
          depthPointsInBox++;
        }
      }
    }
    point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
  println(depthPointsInBox);
  // set the box color's transparency
  // 0 is transparent, 1000 points is fully opaque red
  float boxAlpha = map(depthPointsInBox, 0, 1000, 0,
255);
  translate(boxCenter.x, boxCenter.y, boxCenter.z);
  // the fourth argument to fill() is "alpha"
  // it determines the color's opacity
  // we set it based on the number of points
  fill(255, 0, 0, boxAlpha);
  stroke(255, 0, 0);
  box(boxSize);
}
```

3

```
// use keys to control zoom
// up-arrow zooms in
// down arrow zooms out
// s gets passed to scale() in draw()
void keyPressed(){
  if(keyCode == 38){ // flèche vers le haut
  s = s + 0.01;
  }
  if(keyCode == 40){//flèche vers le bas
  s = s - 0.01;
  }
}
void mousePressed(){
  save("touchedPoint.png");
}
```

IV. AFFICHAGE ET MANIPULATION D'OBJETS EN 3D

Projet : Virtual Kinect (p157)

4.1. Installation de la librairie et 1er programme

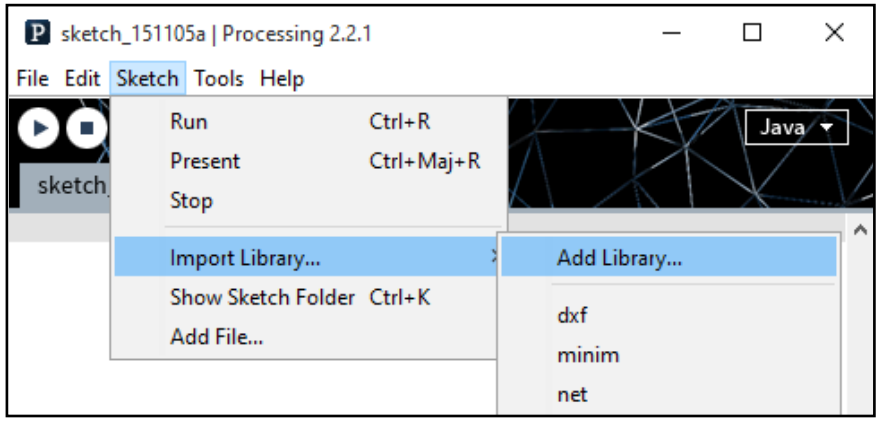
Objectif: Afficher un objet 3D préalablement conçu, l'afficher et le manipuler en 3D

➡ **Télécharger la librairie « OBJLoader »**

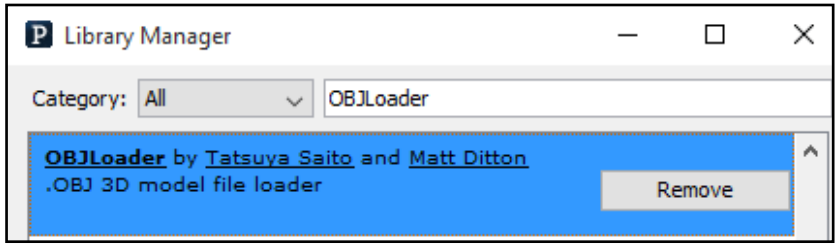
↳ *http://processing.org/reference/libraries*

↳ *http://co de.google.com/p/saitoobjloader*

↳



↳



Télécharger le modèle 3D de la kinect

http://makingthingssee.com/assets/kinect_model.zip

WW2 and ballistics gave us digital computers. Cold War decentralisation gave us the Internet. Terrorism and mass surveillance: Kinect.

— Matt Webb

3D vision with Kinect, Processing, Arduino, and MakerBot

by Greg Borenstein

Welcome to the Vision Revolution. With the Microsoft Kinect you can now use 3D computer vision technology to make 3D scans of people and objects, detect gestures and track people's bodies, and build interactive applications of all kinds. This hands-on guide provides all the technical and conceptual information you need to build cool applications with Kinect, using the Processing programming language, the Arduino microcontroller, and the MakerBot 3D printer.

Resources

Read the Table of Contents
See all the code samples on Github
Making Things See Flickr Set
Making Things See videos on Vimeo
Download the assets for Chapter 3
Get the SimpleOpenNI Processing library

O'REILLY

Cliquer sur ce lien

Télécharger le sketch et placer dans le dossier « data » le fichier « kinect.obj »

WW2 and ballistics gave us digital computers. Cold War decentralisation gave us the Internet. Terrorism and mass surveillance: Kinect.

— Matt Webb

3D vision with Kinect, Processing, Arduino, and MakerBot

by Greg Borenstein

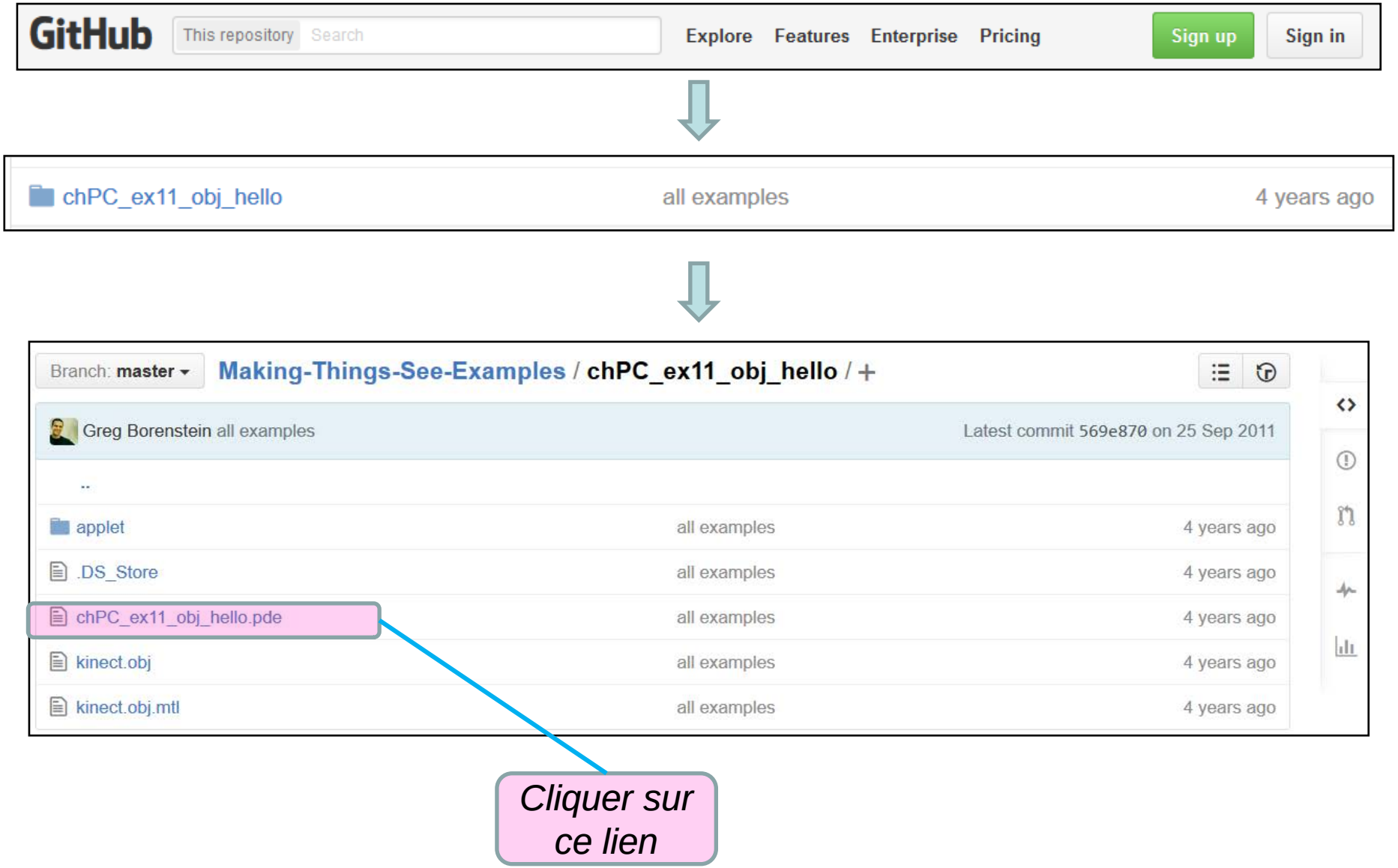
Welcome to the Vision Revolution. With the Microsoft Kinect you can now use 3D computer vision technology to make 3D scans of people and objects, detect gestures and track people's bodies, and build interactive applications of all kinds. This hands-on guide provides all the technical and conceptual information you need to build cool applications with Kinect, using the Processing programming language, the Arduino microcontroller, and the MakerBot 3D printer.

Resources

Read the Table of Contents
See all the code samples on Github
Making Things See Flickr Set
Making Things See videos on Vimeo
Download the assets for Chapter 3
Get the SimpleOpenNI Processing library

Cliquer sur ce lien

O'REILLY


GitHub
This repository
Search
Explore
Features
Enterprise
Pricing
Sign up
Sign in
chPC_ex11_obj_hello
all examples
4 years ago
Branch: master
Making-Things-See-Examples / chPC_ex11_obj_hello / +
Greg Borenstein all examples
Latest commit 569e870 on 25 Sep 2011
..
applet
all examples
4 years ago
.DS_Store
all examples
4 years ago
chPC_ex11_obj_hello.pde
all examples
4 years ago
kinect.obj
all examples
4 years ago
kinect.obj.mtl
all examples
4 years ago
Cliquer sur ce lien

Copier le code

```
import processing.opengl.*;
import saito.objloader.*;

// declare an OBJModel object
OBJModel model;

float rotateX;
float rotateY;

void setup() {
  size(640, 480, OPENGL);

  // load the model file
  // use triangles as the basic geometry
  model = new OBJModel(this, "kinect.obj", "relative", TRIANGLES);

  // tell the model to translate itself
  // to be centered at 0,0
  model.translateToCenter();
  noStroke();
}

void draw() {
  background(255);

  // turn on the lights
  lights();

  translate(width/2, height/2, 0);

  rotateX(rotateY);
  rotateY(rotateX);

  // tell the model to draw itself
  model.draw();
}

void mouseDragged() {
  rotateX += (mouseX - pmouseX) * 0.01;
  rotateY -= (mouseY - pmouseY) * 0.01;
}
```

Commentaires du code

```
import processing.opengl.*;
import saito.objloader.*;

// declare an OBJModel object
OBJModel model;

float rotateX;
float rotateY;

void setup() {
  size(640, 480, OPENGL);

  // load the model file
  // use triangles as the basic geometry
  model = new OBJModel(this, "kinect.obj", "relative", TRIANGLES);

  // tell the model to translate itself
  // to be centered at 0,0
  model.translateToCenter();
  noStroke();
}
```

Librairies:
OpenGL (calcul d'image 3D)
OBJLoader (manipulation d'objet 3D)

Déclaration d'une variable globale kinect de type OBJModel

Instanciation de l'objet « model »: Dans le constructeur, les paramètres 2 (« kinect.obj ») et 4 (« TRIANGLE ») sont les plus importants

Références sur la classe « OBJModel »

3D

» proscene
by Jean Pierre Charalambos
Library that eases the creation of interactive scenes.

» HE_Mesh 2014
by Frederik Vanhoutte
HE_Mesh is an implementation of a half-edge datastructure for manipulating 3D meshes.

» PeasyCam
by Jonathan Feinberg
A mouse driven camera-control library for 3D sketches.

» Picking
by Nicolas Clavaud
Pick an object in a 3D scene easily.

» OCD: Obsessive Camera Direction
by Kristian Damkjer
The Obsessive Camera Direction (OCD) library allows intuitive control and creation of Processing viewport Cameras.

» Patchy
by Jonathan Feinberg
Patchy provides an easy-to-use bicubic patch for 3D Processing sketches.

» OBJLoader
by Tatsuya Saito and Matt Ditton
.OBJ 3D model file loader

» Shapes 3D
by Peter Lager
3D Shape creation and display made easy.

» SimpleOpenNI
by Max Rheiner
A simple wrapper for OpenNI(Kinect-Library). Before you can use SimpleOpenNI you have to install OpenNI.

» iGeo
by Satoru Sugihara
3D geometry library with packages of NURBS geometry, polygon mesh geometry, vector math, 3D display and navigation, 3D data file I/O and agent-based 3D geometry modeling.

» planetarium
by Andres Colubri
This library provides a renderer to project 3D scenes on a full dome.

Cliquer sur ce lien

Reference

This library contains a class for loading/rendering a .OBJ file (OBJModel) and a class for accessing each vertex (Vertex) which can be primarily used to addressing vertexes in the model file and transform it.

For a complete list of documentation go and check the site here.

Cliquer sur ce lien

All Classes

BoundingBox
Debug
Face
Group
Material
OBJModel
Segment

Package \ Class \ Tree \ Deprecated \ Index \ Help \

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package saito.objloader

Class Summary	
BoundingBox	
Debug	
Face	
Group	
Material	
OBJModel	
Segment	

Cliquer sur ce lien

Package \ Class \ Tree \ Deprecated \ Index \ Help \

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Références sur la classe « OBJModel » sont sur cette page:

http://thequietvoid.com/client/objloader/reference/index.html

saito.objloader
Class OBJModel

```
java.lang.Object
  └ saito.objloader.OBJModel
```

```
public class OBJModel
extends java.lang.Object
```

Author:
matt

Field Summary	
static java.lang.String	ABSOLUTE
Debug	debug
static java.lang.String	RELATIVE

Constructor Summary
OBJModel(PApplet _parent) Class Constructor to setup an empty obj model
OBJModel(PApplet _parent, java.lang.String _filename) Class Constructor, loads the file from the data directory
OBJModel(PApplet _parent, java.lang.String _fileName, int _shapeMode) Class Constructor, loads the string as an obj from the data directory.
OBJModel(PApplet _parent, java.lang.String _fileName, java.lang.String _texturePathMode)

Commentaires du code

```
import processing.opengl.*;
import saito.objloader.*;

// declare an OBJModel object
OBJModel model;

float rotateX;
float rotateY;

void setup() {
  size(640, 480, OPENGL);

  // load the model file
  // use triangles as the basic geometry
  model = new OBJModel(this, "kinect.obj", "relative", TRIANGLES);

  // tell the model to translate itself
  // to be centered at 0,0
  model.translateToCenter();
  noStroke();
}
```

La méthode « translateToCenter() » initialise les coordonnées internes du model 3D de sorte que son centre géométrique soit à l'origine de la fenêtre d'affichage (0,0).
Cette manipulation est nécessaire pour faciliter les rotations et les translations du modèle…

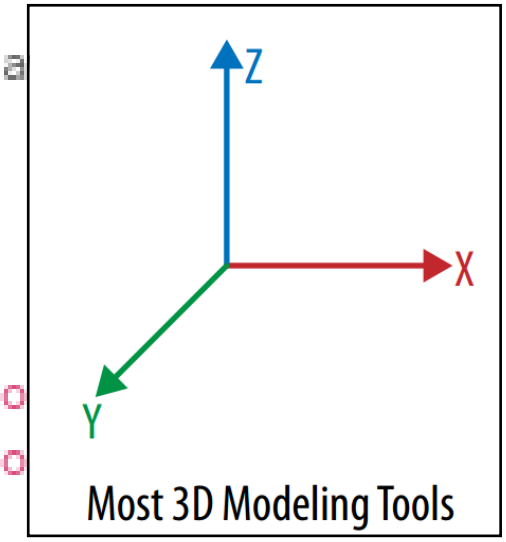
Commentaires du code

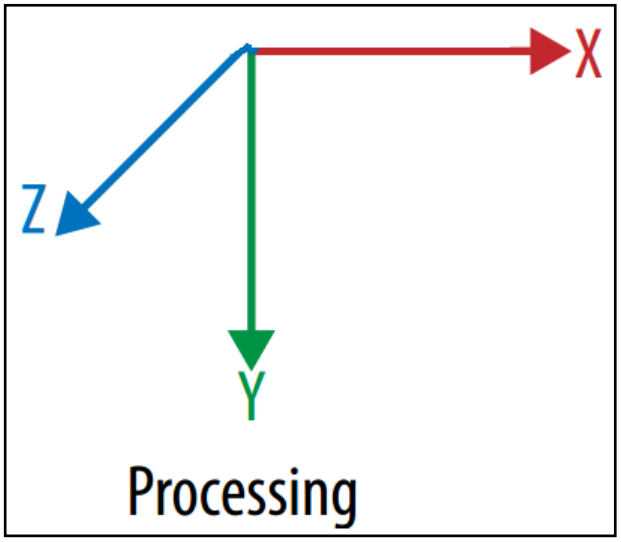
```
void draw() {
  background(255);

  // turn on the lights
  lights();

  translate(width/2, height/2, 0);

  rotateX(rotateY);
  rotateY(rotateX);

  // tell the model to dra
  model.draw();
}

void mouseDragged() {
  rotateX += (mouseX - pmo
  rotateY -= (mouseY - pmo
}
```

Allumage de la lumière pour faciliter la vue en 3D

Translation de l'origine (et donc du model 3D !!) au centre de la fenêtre d'affichage

Les axes du modèle 3D et ceux de Processing ne sont pas les mêmes…





Commentaires du code

```
void draw() {
  background(255);

  // turn on the lights
  lights();

  translate(width/2, height/2, 0);

  rotateX(rotateY);
  rotateY(rotateX);

  // tell the model to draw itself
  model.draw();
}

void mouseDragged() {
  rotateX += (mouseX - pmouseX) * 0.01;
  rotateY -= (mouseY - pmouseY) * 0.01;
}
```

Dessin du modèle après avoir subi l'ensemble des rotations rotateX() et rotateY() …

Commentaires du code

```
void draw() {
  background(255);

  // turn on the lights
  lights();

  translate(width/2, height/2, 0);

  rotateX(rotateY);
  rotateY(rotateX);

  // tell the model to draw itself
  model.draw();
}

void mouseDragged() {
  rotateX += (mouseX - pmouseX) * 0.01;
  rotateY -= (mouseY - pmouseY) * 0.01;
}
```

- Initialement rotateX et rotateY vallent 0.
- pmouseX et pmouseY: position de la sourie dans l'ancienne fenêtre d'affichage.
- mouseX et mouseY: position de la sourie dans l'actuelle fenêtre d'affichage;

mouseX et mouseY ne sont actualisées qu'une fois par frame alors que mouseEvent() actualise leur valeur à chaque nouvel événements… (voir la référence de pmouseX)

Pour comprendre l'affichage en 3D copier le code suivant à la fin du sketch et taper sur n'importe quelle touche du clavier:

```
boolean drawLines = false;

void keyPressed(){
 if(drawLines){
 model.shapeMode(LINES);
 stroke(0);
 } else {
 model.shapeMode(TRIANGLES);
 noStroke();
 }
 drawLines = !drawLines;
}
```

```
boolean drawLines = false;

void keyPressed(){
 if(drawLines){
 model.shapeMode(LINES);
 stroke(0);
 } else {
 model.shapeMode(TRIANGLES);
 noStroke();
 }
 drawLines = !drawLines;
}
```

4.2. Insertion du modèle 3D dans le nuage de points (p166)

Objectif: Afficher le modèle 3D de la kinect dans l'image du nuage de points à la même position que la kinect réelle…

Problème:

- Positionner la kinect à l'origine des axes dans les nuage de point a été dessiné.
- Pour cela, prendre en compte les différentes rotations, translations et homothéties (effets d'échelle) qu'il a été nécessaire d'effectuer pour positionner le nuage de points au centre de la fenêtre d'affichage pour le manipuler…
- Le modèle 3D de la kinect doit subir les mêmes transformations géométriques avant d'être dessinées

Solution: Dessiner le modèle après avoir effectué les transformations…

```
import processing.opengl.*;
import SimpleOpenNI.*;
import saito.objloader.*;

SimpleOpenNI kinect;
OBJModel model;

float s = 1;

void setup() {
  size(1024, 768, OPENGL);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();

  model = new OBJModel(this, "kinect.obj", "relative", TRIANGLES);
  model.translateToCenter();
  noStroke();
}

void draw() {
  background(0);
  kinect.update();

  translate(width/2, height/2, -1000);
  rotateX(radians(180));

  translate(0, 0, 1400);
  rotateY(radians(map(mouseX, 0, width, -180, 180)));

  translate(0, 0, s*-1000);
  scale(s);

  lights();
  noStroke();

  // isolate model transformations
  pushMatrix();
    // adjust for default orientation
    // of the model
    rotateX(radians(-90));
    rotateZ(radians(180));
    model.draw();
  popMatrix();

  stroke(255);

  PVector[] depthPoints = kinect.depthMapRealWorld();

  for (int i = 0; i < depthPoints.length; i+=10) {
    PVector currentPoint = depthPoints[i];
    point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
}

void keyPressed() {
  if (keyCode == 38) {
    s = s + 0.01;
  }
  if (keyCode == 40) {
    s = s - 0.01;
  }
}
```

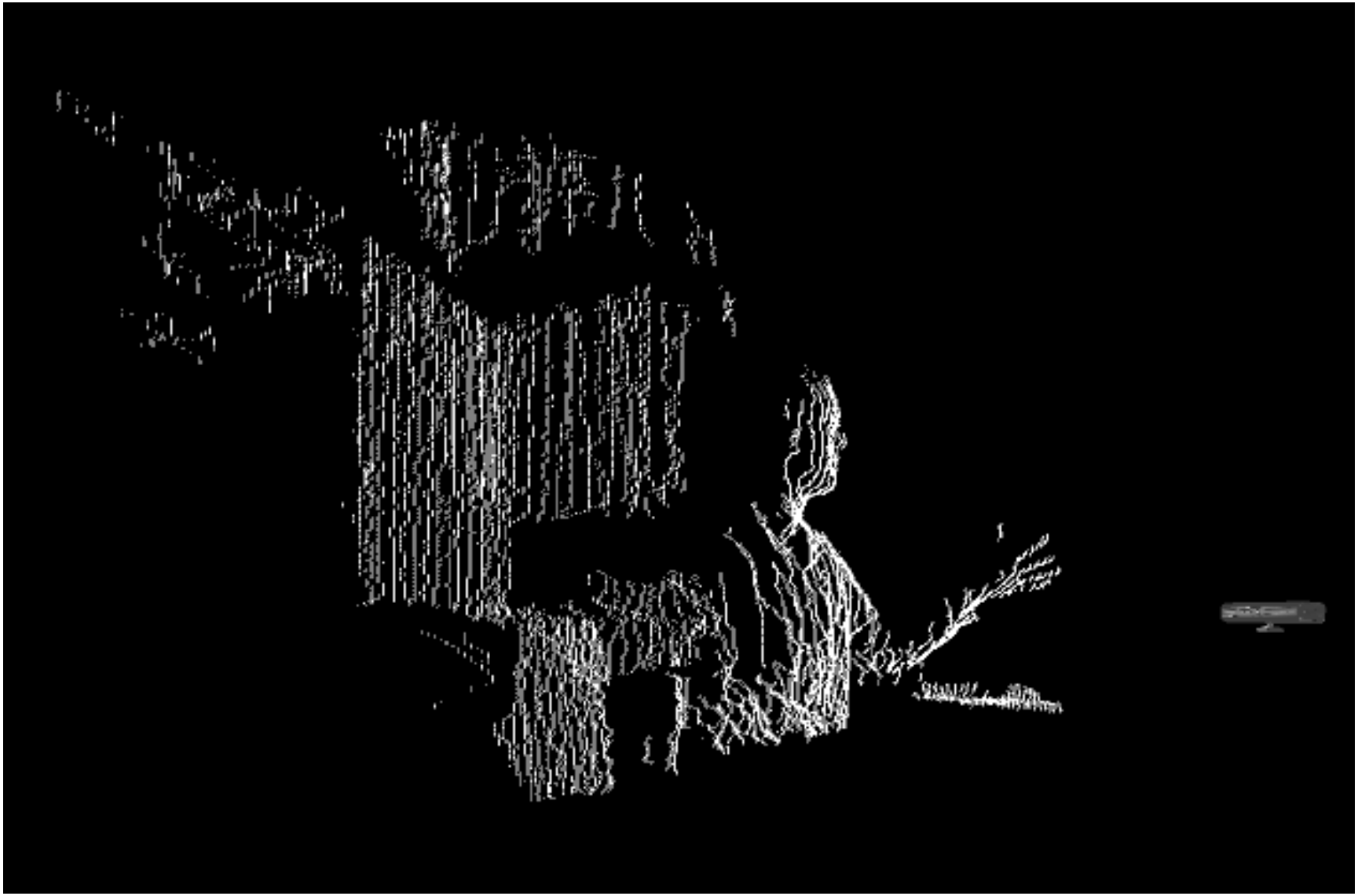
Copier le code:

https://github.com/atduskgreg/Making-Things-See-Examples/tree/master/chPC_ex12_obj_in_point_cloud

Bien insérer le fichier « kinect.obj » dossier « data » du sketch

Commentaires: voir p168/169

Résultat:



CHAPITRE 4

TRAVAILLER SUR LES DONNÉES DU SQUELETTE

(SKELETON DATA)

IV. TRAVAILLER SUR LES DONNÉES DU SQUELETTE

4.1. Présentation – les possibilités de la librairie OpenNI

- **User-tracking data** : utilisation des données de tracking
  - Détection/tracking d'un utilisateur
- **Sketch interactif** qui répond aux action de l'utilisateur
  - … par utilisation des données de l'image en profondeur
- Détection des « **articulations** » de chaque utilisateur !!
  - Tête, cou, épaules, coudes, main, torse, hanches, genoux, pieds…
- **Techniquement plus difficile:**
  - **Calcul vectoriel (class PVector)**

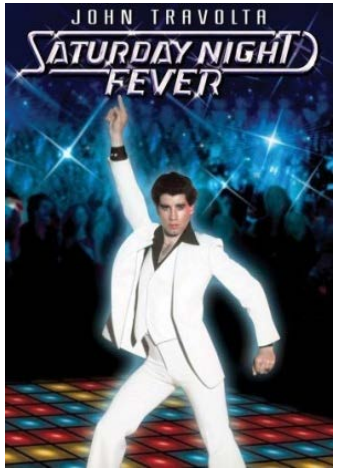
- **Calibration du tracking**
  - Nécessité pour chaque utilisateur de calibrer le tracking
    - Permet l'accessibilité des articulations

- **Notifications de tracking de SimpleOpenNI**
  - Permets de vérifier/suivre le bon fonctionnement du tracking
    - Détection d'un utilisateur
    - Début du tracking
    - Perte de l'utilisateur
    - Etc .

- **Dessin d’un bonhomme « en bâton »**
  - A partir des articulations détectées
- **Analyse des positions et du mouvement des utilisateurs**
  - Mesure de distances entre articulations
  - Mesure d’angles entre les membres du corps
    - Clé pour comprendre les positions d’un individu en 3D …
    - Nécessité du calcul vectoriel
- **Analyse du contenu d’une scène**
  - Analyse de la scène pour la **suppression du fond**
  - Analyse de posture et traking de la main
  - Détection du centre de masse d’un individu

Les différents projets abordés

- **Retour d'expérience et correction des postures dans des exercices physiques**
  - Gestes correctes dans les mouvements
  - Rééducation / kinésie thérapie
  - Comparaison entre le mouvement juste préenregistré et le mouvement réel…
- **Détection de position dans une chorégraphie**
  - Saturday Night Fever



- **Création d'applications interactive contrôlées par le corps**

4.2. 1er programme: Détection et tracking de la main

➡ Récupérer le code sur :

https://github.com/atduskgreg/Making-Things-See-Examples/blob/master/chSK_ex01_one_joint/chSK_ex01_one_joint.pde

1

```
import SimpleOpenNI.*;
SimpleOpenNI  kinect;

void setup() {
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();

  // turn on user tracking
  kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);

  size(640, 480);
  fill(255,0,0);
}
```

3

```
// user-tracking callbacks!
void onNewUser(int userId) {
  println("start pose detection");
  kinect.startPoseDetection("Psi", userId);
}

void onEndCalibration(int userId, boolean successful) {
  if (successful) {
    println("  User calibrated !!!");
    kinect.startTrackingSkeleton(userId);
  } else {
    println("  Failed to calibrate user !!!");
    kinect.startPoseDetection("Psi", userId);
  }
}

void onStartPose(String pose, int userId) {
  println("Started pose for user");
  kinect.stopPoseDetection(userId);
  kinect.requestCalibrationSkeleton(userId, true);
}
```

2

```
void draw() {
  kinect.update();
  PImage depth = kinect.depthImage();
  image(depth, 0, 0);

  // make a vector of ints to store the list of users
  IntVector userList = new IntVector();
  // write the list of detected users
  // into our vector
  kinect.getUsers(userList);

  // if we found any users
  if (userList.size() > 0) {
    // get the first user
    int userId = userList.get(0);

    // if we're successfully calibrated
    if ( kinect.isTrackingSkeleton(userId)) {
      // make a vector to store the left hand
      PVector rightHand = new PVector();
      // put the position of the left hand into that vector
      kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_HAND, rightHand);

      // convert the detected hand position
      // to "projective" coordinates
      // that will match the depth image
      PVector convertedRightHand = new PVector();
      kinect.convertRealWorldToProjective(rightHand, convertedRightHand);
      // and display it
      ellipse(convertedRightHand.x, convertedRightHand.y, 10, 10);
    }
  }
}
```

```
import SimpleOpenNI.*;
SimpleOpenNI  kinect;

void setup() {
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();

  // turn on user tracking
  kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);

  size(640, 480);
  fill(255,0,0);
}
void draw() {
  kinect.update();
  PImage depth = kinect.depthImage();
  image(depth, 0, 0);

  // make a vector of ints to store the list of users
  IntVector userList = new IntVector();
  // write the list of detected users
  // into our vector
  kinect.getUsers(userList);

  // if we found any users
  if (userList.size() > 0) {
    // get the first user
    int userId = userList.get(0);

    // if we're successfully calibrated
    if ( kinect.isTrackingSkeleton(userId)) {
      // make a vector to store the left hand
      PVector rightHand = new PVector();
      // put the position of the left hand into that vector
      kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_HAND, rightHand);

      // convert the detected hand position
      // to "projective" coordinates
      // that will match the depth image
      PVector convertedRightHand = new PVector();
      kinect.convertRealWorldToProjective(rightHand, convertedRightHand);
      // and display it
      ellipse(convertedRightHand.x, convertedRightHand.y, 10, 10);
    }
  }
}
```

Copier le code

```
// user-tracking callbacks!
void onNewUser(int userId) {
  println("start pose detection");
  kinect.startPoseDetection("Psi", userId);
}

void onEndCalibration(int userId, boolean successful) {
  if (successful) {
    println("  User calibrated !!!");
    kinect.startTrackingSkeleton(userId);
  } else {
    println("  Failed to calibrate user !!!");
    kinect.startPoseDetection("Psi", userId);
  }
}

void onStartPose(String pose, int userId) {
  println("Started pose for user");
  kinect.stopPoseDetection(userId);
  kinect.requestCalibrationSkeleton(userId, true);
}
```

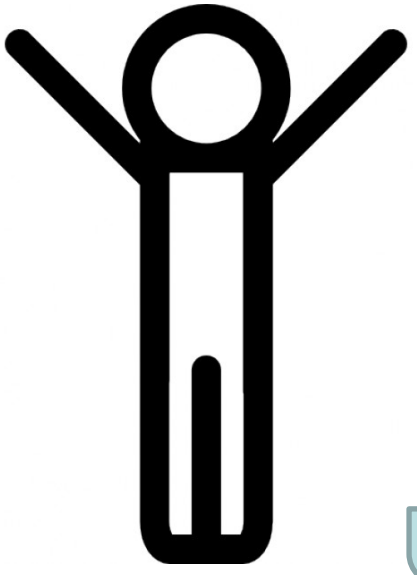
Pb de compatibilité de version de simpleOpenNI:

https://code.google.com/p/simple-openni/issues/detail?id=76

Lancer directement :
Processing/libraries/SimpleOpenNI/examples/OpenNi/User3d

- Description rapide du programme:
  - Détection de l'utilisateur
  - Calibration de sa position
  - Utilisation des données de position des articulations (joints) pour le tracking de la main
  - Dessin d'un cercle sur la position de la main droite…

Remarques concernant la calibration



La calibration doit s'effectuer dans une **position connue**:

- **Debout**
- **Les jambes l'une contre l'autre**
- **Les bras levés au dessus des épaules au niveau de la tête**

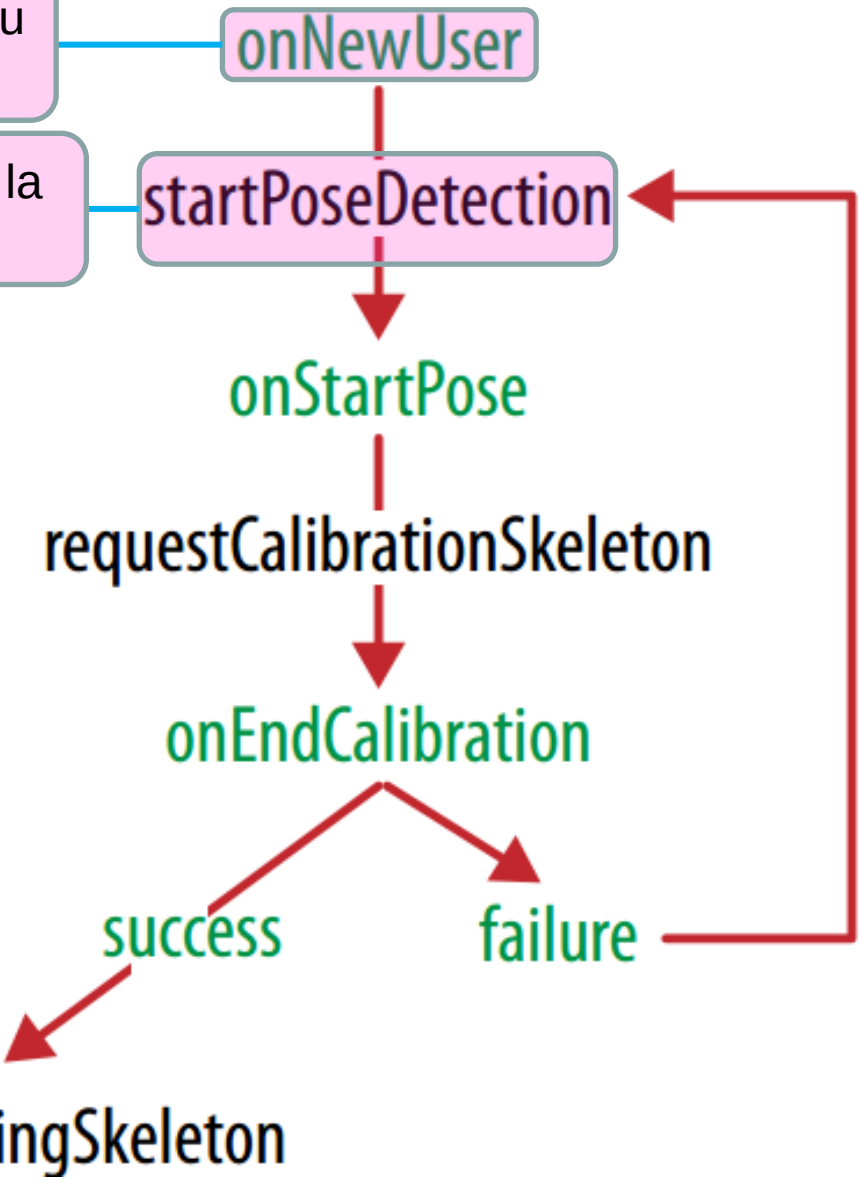
  - La pause « Y » (« Psi » pause)

Les différentes étapes de la calibration : système bouclé

OpenNI détecte la présence d'un individu et lance la méthode « **onNewUser** »

Initialisation du tracking par l'appel de la méthode « **startPoseDetection** »

Test de la « pose Y » pour tous les utilisateurs



Les différentes étapes de la calibration : système bouclé