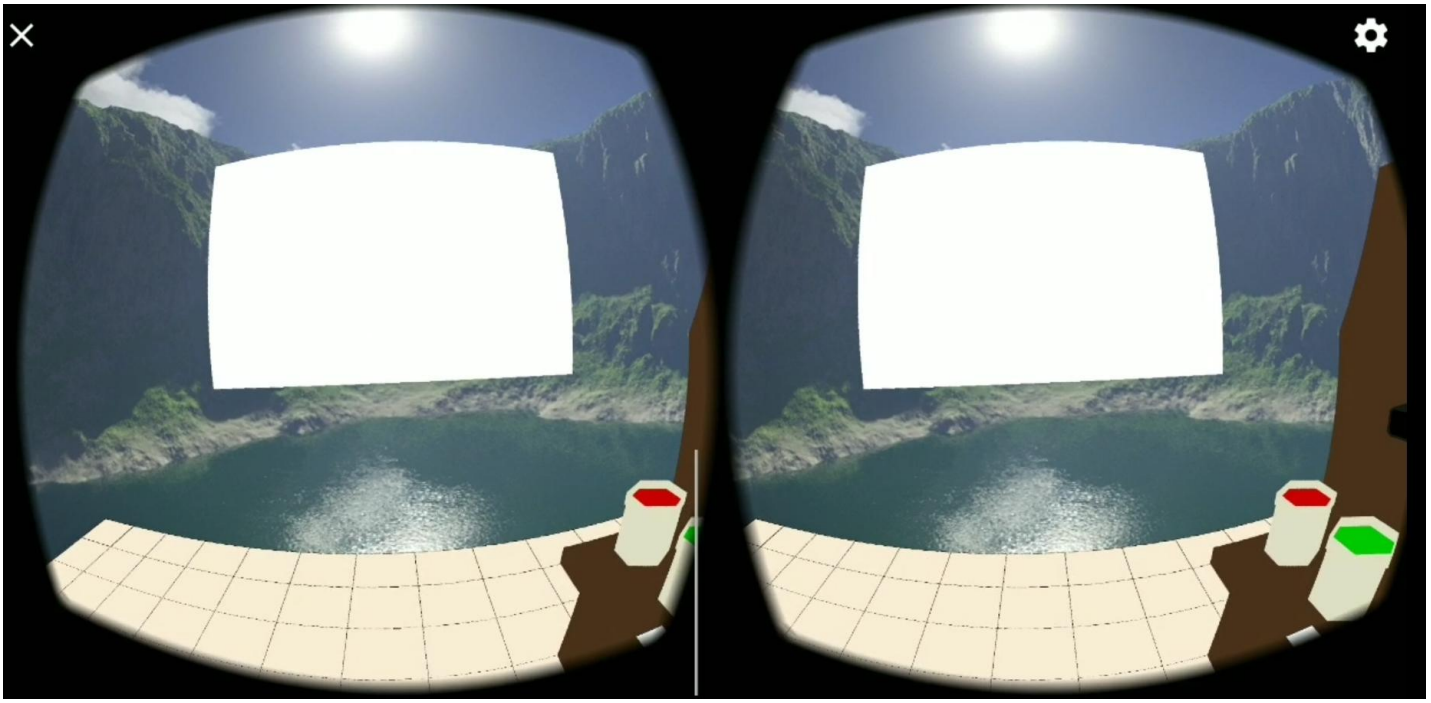
scribbles in VR

“Un simulation de peinture en réalité virtuelle faite maison”



Un projet réalisé par :
Louis Leger & Ixil Miniussi

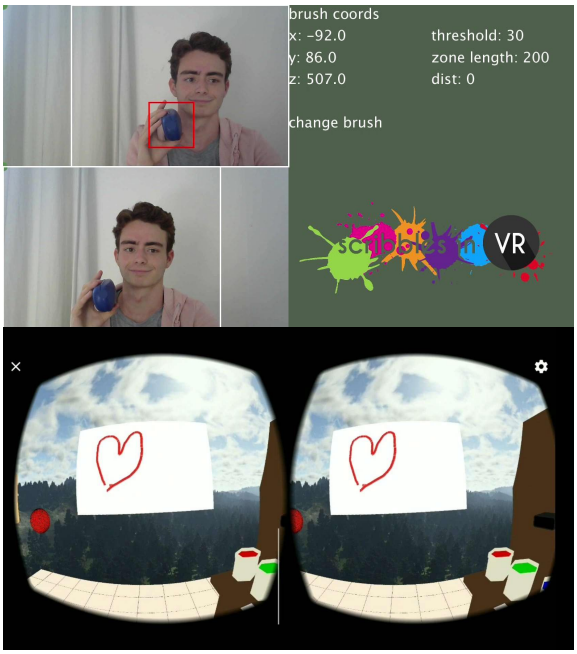
Table des matières

I) Introduction et présentation générale

L'idée et les objectifs du projet :

L'idée de ce projet est née de notre volonté d'explorer Processing en 3D/réalité virtuelle, le networking entre dispositifs et le traitement d'images. Nous souhaitions le faire avec les moyens à notre disposition. Nous avons tout d'abord conçu un casque de réalité virtuelle "fait maison" à l'aide de deux caméras. Puis, en avançant dans sa conception, nous avons trouvé l'idée de faire un programme de peinture en réalité virtuelle.

Présentation du projet final :



vidéo de présentation :
https://www.youtube.com/watch?v=_4Z0W0Z2gyc&feature=youtu.be

Fonctionnalitées du programme (Quoi?):

La personne souhaitant peindre virtuellement doit s'asseoir devant un PC muni de deux caméras et mettre un casque de réalité virtuelle, constitué principalement d'un téléphone, d'un "google cardboard" et d'une lampe recouverte d'un plastique vert. Puis, il se munit d'une lampe-torche recouverte d'un plastique rouge. Le casque le fait entrer dans un univers virtuel, qui suit les mouvements de sa tête, et la lampe torche sert de contrôle et de pinceau pour peindre dans cet univers.

La lampe torche est reflétée dans le monde virtuel par une boule de couleur qui suit les mouvements de la main. Le joueur peut donc déplacer cette boule pour accéder au menu, ou l'utiliser comme pinceau pour peindre sur une toile imaginaire. Le menu permet de commencer une partie, dans laquelle le joueur peut, à l'aide de plusieurs boutons:
- commencer un nouveau dessin
- effacer un dessin
-choisir un univers virtuel (parmi les 8 prédéfinis)
-choisir un sujet de dessin (dix par univers)

Le joueur peut aussi exporter son dessin sur le pc dans un format d'image ".jpg" ainsi que changer de couleur de peinture en déplaçant la boule dans un seau de peinture virtuel (4 seaux : noir, rouge, bleu et vert). La boule change alors de couleur selon le seau choisit.

Résumé du fonctionnement (Comment?) :

En utilisant deux caméras, placées côte à côte, nous parvenons à détecter les coordonnées de 2 zones d'intérêts colorées. En s'appuyant sur la différence entre les coordonnées x des pixels renvoyés par chaque caméra et sur la trigonométrie, nous pouvons retrouver les coordonnées de ces pixels dans l'espace tridimensionnel. Ces coordonnées sont ensuite envoyées sur un téléphone avec le format de transmission "Open Sound Control". Elles sont utilisées pour dessiner sur une toile imaginaire et pour modifier la position de la tête du joueur dans l'univers virtuel, le tout dans un téléphone (placé dans une boîte de google cardboard).

La répartition du travail (Qui?):

Il y a deux programmes principaux qui ont permis la réalisation de ce projet: un pour l'ordinateur et un autre pour le téléphone. Louis a développé le programme du téléphone et Ixil a programmé celui de l'ordinateur avec les caméras. Toutefois, au fur et à mesure, de l'avancement de la programmation, les difficultées rencontrées ont été partagées afin de trouver les solutions par une réflexion commune. Par ailleurs, certains développements spécifiques ont pu être développés par chacun en dehors de la répartition des tâches initiales. Ainsi, Louis s'est aussi occupé de la visualisation du programme de l'ordinateur et a apporté quelques améliorations à certaines parties du programme. Ixil , quant à lui, s'est occupé du mouvement de la tête et de quelques aspects de l'environnement dans le programme du téléphone.
Le projet a aussi constitué un véritable travail en équipe où chacun a apporté ses idées et solutions au fur et à mesure de son avancement.

II) Explication des aspects principales du projet

Nous avons choisi de nous concentrer sur les aspects qui nous paraissent les plus importants du programme du pc et du téléphone, nommé respectivement : paint_pc et paint_vr.

A) "paint pc"

1. Détection de zones d'intérêt (Blob detection)

En programmation, la détection de zones d'intérêt (ou blob detection) est un ensemble de méthodes dédiées à la détection et mise en avant de régions spécifiques dans une image numérique. Dans notre cas, nous cherchions un moyen d'extraire la position, sous la forme de coordonnées bidimensionnelles, de la main ainsi que de la tête du joueur, afin de les implémenter dans le jeu. Pour ce faire, nous avons établi un code couleur, le joueur tiendrai dans sa main un bâton rouge, et sur sa tête serait accroché une balle verte (2 couleurs primaires, peu communes dans nos environnements). Pour détecter le bâton rouge du joueur, nous partons du principe que la majorité des pixels rouges de l'image seront sur le bâton. Il nous suffit alors, pour chaque image de la vidéo, de trouver les coordonnées de chaque pixel rouge sur l'image, et d'en faire la moyenne. On répète ensuite le processus pour la couleur verte.

```
int pos = i + (j * cam.width);
color pixel = cam.pixels[pos];
```

Pour cela, nous transformons l'image en un tableau de couleurs *[paint_pc l.388]* avec cam.pixels[pos].
cam correspond à l'image actuelle prise par la vidéo, et .pixels[pos], transforme cam en une liste unidimensionnelle de couleurs, que l'on peut explorer à travers pos. Par exemple, avec notre image cam :

nous avons cam.pixels[] = ((255, 0, 0), (0, 0, 0), (0, 255, 0), (0, 0, 255), (0, 0, 255), (255, 255, 255);
et ainsi cam.pixels[1] = (0, 0, 0); On remarque que .pixels[] retranscrit notre image ligne par ligne, d'où int pos = i + (j * cam.width); *[paint_pc l.387{* où i correspond aux coordonnées x du pixel, et j aux coordonnées y. Dans notre exemple, le pixel vert se trouve à x=2 et y=0 (pour l'ordinateur, la première valeure d'une liste correspond à 0 et non à 1),

avec cam.width = 3, on retrouve bel et bien cam.pixels[2 + (0 * 3)] = (0, 255, 0);.
Nous pouvons alors naviguer sur tous les pixels d'une image. Nous les comparons tous *(à travers deux boucles for() |paint_pc l.383/384{ balayant les pixels de l'image colonne par colonne)* à la couleur recherchée à l'aide du théorème de Pythagore.

(Il existe dans processing une commande dist( ) qui permet de trouver la distance entre deux couleurs, en revanche, cette commande opère une racine carrée qui prend plus de mémoire que nécessaire. Nous préférons faire cette opération nous même à travers :

```
float d(float r, float g, float b, float r1, float g1, float b1) {

  return (r1 - r) * (r1 - r) + (g1 - g) * (g1 - g) + (b1 - b) * (b1 - b);
}
```

c'est à dire en élevant threshold au carré.)
Si cette couleur vérifie la condition, nous conservons ses coordonnées pour ensuite en faire la moyenne, n_pixels s'incrémente pour faire le compte du nombre de coordonnées gardées en mémoire.

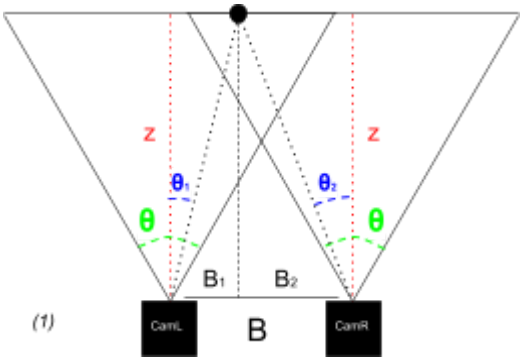
```
if (unit < threshold*threshold) {

  total_x += i;
  total_y += j;
  n_pixels++;
}
```

Nous considérons alors les coordonnées x de l'objet comme étant: total_x/n_pixels, et ses coordonnées y comme étant: total_y/n_pixels.[1] Nous pouvons maintenant situer dans l'image la main et la tête du joueur.

[1] En ce qui concerne la liste "zone[]" voir III) Optimisations. Pour la suite de l'explication on traitera zone[0 + ...] = 0, zone[1 + ...] = 0, zone[2 + ...] = 1280 et zone[3 + ...] = 720.

2. Calculs des coordonnées

Dans l'optique de créer un jeu de réalité virtuelle, où le joueur et son pinceau se déplacent dans un environnement tridimensionnel, nous avions besoin de situer le joueur dans son environnement 3d, et ce, seulement à partir de notre détection de zone d'intérêt. Pour ce, nous avons recherché la "mesure de distance par stéréo-vision". Similaire au fonctionnement de nos yeux, cette méthode permet de déterminer la distance entre nos caméras et des zones d'intérêts, en installant 2 caméras similaires, l'une à côté de l'autre. Nous avons alors acheté une seconde logitech c270 (webcam), identique au modèle que Ixil possédait déjà. Notre programme effectue maintenant la détection de la tête et du pinceau du joueur simultanément sur les deux caméras. Voici comment nous procédons :



(1) Par construction et par trigonométrie, on peut établir que :

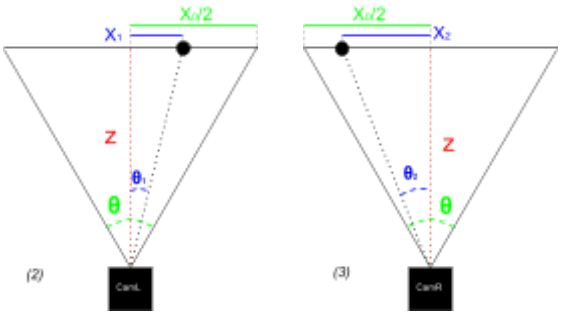
$B_1 = z \times tan(\theta_1)$

et de la même façon que :$B_2 = z \times tan(\theta_2)$

Sachant que$B = B_1 + B_2$, on a alors:

$B = z \times tan(\theta_1) + z \times tan(\theta_2)$

$\Leftrightarrow z = \frac{B}{tan(\theta_1)+tan(\theta_2)}$



(2)

$$\frac{x_1}{\frac{x_0}{2}} = \frac{tan(\theta_1)}{tan(\frac{\theta}{2})} \Leftrightarrow tan(\theta_1) = \frac{tan(\frac{\theta}{2}) \times x_1}{\frac{x_0}{2}}$$

(3)

$$\frac{x_2}{\frac{x_0}{2}} = \frac{tan(\theta_2)}{tan(\frac{\theta}{2})} \Leftrightarrow tan(\theta_2) = \frac{tan(\frac{\theta}{2}) \times x_2}{\frac{x_0}{2}}$$

et ainsi :

$$z = \frac{B}{\frac{tan(\frac{\theta}{2})(x_1+x_2)}{\frac{x_0}{2}}} \Leftrightarrow z = \frac{x_0 \times B}{2 \times tan(\frac{\theta}{2})(x_1+x_2)}$$

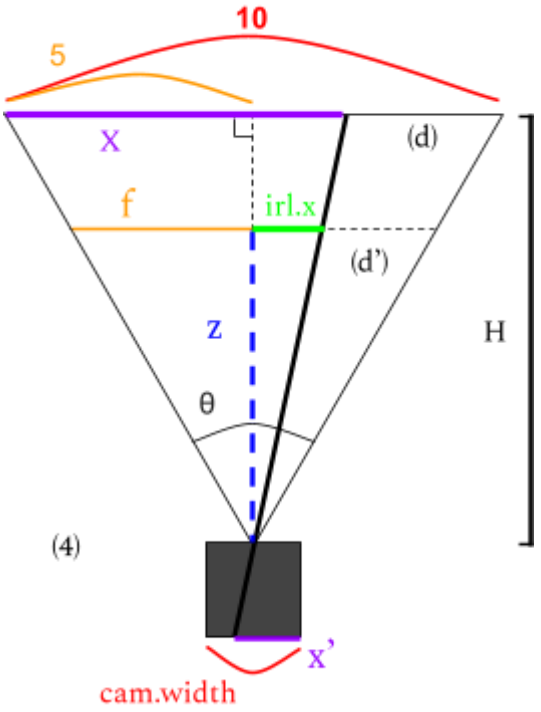
à noter:
dans notre cas, nous préférons utiliser

$$x_1' = x_1 + \frac{x_0}{2} \text{ et } x_2' = -x_2 + \frac{x_0}{2}$$

car ces valeurs correspondent directement aux coordonnées de la zone d'intérêt données par notre fonction detect();
Nous utiliserons donc

$$z = \left| \frac{x_0 \times B}{2 \times tan(\frac{\theta}{2})(x_1' - x_2')} \right|$$

(z est une distance, ainsi on prend la valeur absolue pour éviter tout problème.)



$$\frac{irl.x}{X-5} = \frac{z}{h} = \frac{f}{5}(1) \quad \frac{X}{10} = \frac{x'}{cam.width}(2)$$

ainsi :

(1) $$irl.x = \frac{(X-5)\times f}{5} = \frac{(X-5)\times tan(\frac{\theta}{2})\times z}{5}$$

(2) $$X = \frac{10\times x'}{cam.width}$$

donc :

$$irl.x = \frac{tan(\frac{\theta}{2})\times z\times(\frac{10\times x'}{cam.width}-5)}{5}$$

de la même façon nous pouvons trouver irl.y, mais il faut prendre en compte le ratio $\frac{largeur}{longueur}$ de la caméra, chez nous, ce ratio vaut $\frac{9}{16}$donc irl.y vaut:

$$irl.y = \frac{tan(\frac{\theta}{2})\times z\times(\frac{10\times Y}{cam.width}-5)}{5}\times\frac{9}{16}$$

Nous retrouvons toutes ces formules dans PVector irl() *|paint_pc l.430|* :

Ainsi nous pouvons maintenant trouver les coordonnées irl.x et irl.y du joueur dans son environnement. irl(0,0,0) étant le centre de la caméra droite.

```
PVector irl(int b, int X, float teta, PVector blob_left, PVector blob_right, PVector current, float easing) {

  PVector blob = current;

  float target_z = abs((b * X)/(2 * tan(teta/2) * (blob_left.x - blob_right.x)));

  float step_z = target_z - blob.z;
  blob.z += int(step_z * easing);

  float target_x = int(((target_z * tan(teta/2)) * ((blob_right.x*10)/video_1.width - 5))/5);

  float step_x = target_x - blob.x;
  blob.x += int(step_x * easing);

  float target_y = int((((target_z * tan(teta/2)) * ((blob_right.y*10)/video_1.height - 5))/5)*9/16);

  float step_y = target_y - blob.y;
  blob.y += int(step_y * easing);


  return blob;
}
```

(4) (d) // (d'), le théorème de Thalès nous donne les égalités suivantes:

Même si nous sommes fiers de notre détection de zones d'intérêt et de nos formules, les appliquer sans contention donne une expérience peu plaisante, où la tête du joueur tremble toujours légèrement, de même pour son pinceau. Nous préférons alors appliquer un léger *easing* à nos valeurs, d'où la présence de step_x, step_y, et step_z.

B) ”paint_vr”

1. L’Envoi/Réception des informations via OSC.

Avec maintenant les bonnes coordonnées du pinceau et de la tête, nous allons pouvoir les envoyer vers le programme sur le téléphone. Pour cela nous avons choisi d’utiliser le format de transmission OSC (Open Sound Control) et sa librairie adaptée sur Processing “oscP5”.
OSC est un protocole réseau qui utilise 2 paramètres pour communiquer, une adresse IP et un seul port de communication, ce qui le rend plus simple que d’autres protocols comme TCP qui utilisent plusieurs ports et ont besoin de plus de 2 paramètres. Les messages de la libraire oscP5, sont soit des *integers* et *floats* (codés sur 32 bits) ou des *strings*.
Dans le programme du pc, Il est nécessaire d’initialiser une variable NetAddress (ici appelé “remoteLocation”) qui prend 2 paramètres : une adresse IP (celle de l’autre dispositif, sur le même réseau WiFi) et un numéro de port. Le NetAddress est un des paramètres pour plusieurs fonctions oscP5 comme osc.send();

```
remoteLocation = new NetAddress("192.168.43.186", 54003);
```

Pour le programme sur téléphone, il faut initialiser un objet oscP5 qui démarre le oscP5, et écoute des messages entrants sur le port 54003.

```
osc = new  OscP5(this, 54003);
```

Ensuite pour l’envoi des informations sur le programme du pc :

```
460 void send() { // un void qui sera appelé dans le draw vers la fin de la boucle.
461
462   OscMessage msg = new OscMessage("ffffff"); //definit une nouvelle variable de message OSC, et précise une "tag"
463   OscMessage msg2 = new OscMessage("fff"); // la "tag" nous sera utile plus tard dans la differenciation des messages
464
465   msg.add(calib(brush.x, "x")); //ajoute les floats de la position de la main calibré
466   msg.add(calib(brush.y, "y"));
467   msg.add(calib(brush.z, "z"));
468
469   msg.add(calib(pbrush_x, "x"));//ajoute les floats de la position de la main dans la boucle draw d'avant calibré
470   msg.add(calib(pbrush_y, "y"));
471   msg.add(calib(pbrush_z, "z"));
472
473   msg2.add((head.x-distance_to_camera.x)/2); // ajoute les floats de la position de la tête dans un nouveau message
474   msg2.add((head.y-distance_to_camera.y)/2);
475   msg2.add((head.z-distance_to_camera.z)/2);
476
477   osc.send(msg, remoteLocation); //envoie le message OSC, sur la remote location, une variable de type NetAddress (libraire oscP5) defini dans le setup()
478   osc.send(msg2, remoteLocation);
479 }
```

Pour la réception des messages, les messages osc entrants sont transmis à la méthode/fonction oscEvent. (comme le mousePressed, ce void n’est appelé que lorsque des paquets d’information sont reçus sur le port attribué dans le setup();.

```
379 void oscEvent(OscMessage msg) {
380
381   if (msg.checkTypetag("ffffff")) { //verifie le "tag" du message reçu et s'il correspond à un message de tag 6 floats "ffffff"
382
383     brush.x = msg.get(0).floatValue(); //réception des coordonnées du pinceau (brush), de type float
384     brush.y = msg.get(1).floatValue();
385     brush.z = msg.get(2).floatValue();
386
387     pbrush.x = msg.get(3).floatValue();//réception des coordonnées du pinceau dans le draw() antérieur (brush), de type float
388     pbrush.y = msg.get(4).floatValue();
389     pbrush.z = msg.get(5).floatValue();
390   }
391
392   if (msg.checkTypetag("fff")) { //verifie le "tag" du message reçu et s'il correspond à un message de tag 3 floats "fff"
393
394     head.x = msg.get(0).floatValue(); //réception des coordonnées de la position de la tête
395     head.y = msg.get(1).floatValue();
396     head.z = msg.get(2).floatValue();
397   }
398 }
```

2. Le programme pour dessiner

Le programme le plus simple pour dessiner sur processing est le suivant :

```
void setup() {
  size(300,300);
}
void draw() {
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```

En revanche il y a trois grand problèmes avec ce programme :

1) nous voulons dessiner avec le pinceau du joueur (et non la souris).
2) le programme superpose les lignes car le fond d'écran est statique (fonction background inexistante).
3) le manque de contrôle pour savoir si on dessine ou pas.

Pour le 1), avec les coordonnées du pinceau (un PVector brush), il suffit de créer des variables pbrush_x/_y/_z qui seront égales au brush.x/.y/.z . Cette égalité sera la dernière action dans le boucle draw().

```
  pbrush_x = brush.x;
  pbrush_y = brush.y;
  pbrush_z = brush.z;
}
```

La solution que nous avons adoptée pour le 2) et 3) est plus compliquée. Nous avons choisi de créer une class strokes (= trait en anglais), avec comme paramètres dans son constructeur les 4 coordonnées 2D de la ligne (x, y, x_end, y_end), l'épaisseur du trait s et sa couleur c, et un void display() pour afficher la ligne.

```
class strokes {

  float x, y, x_end, y_end, s;
  color c;

  strokes(float x_, float y_, float x__, float y__, float w, color c_) {

    x = x_;
    y = y_;
    x_end = x__;
    y_end = y__;
    s = w;
    c = c_;
  }

  void display() {

    strokeWeight(s);
    stroke(c);
    fill(c);

    line(x, y, x_end, y_end);
  }
}
```

Ensuite nous avons créé une liste d'objets (ArrayList) du type strokes appelé canvas (toile en anglais) et dans la boucle draw() nous avons mis une condition qui vérifie si le pinceau (brush) est sur le tableau (avec une fonction on() et les variables x, y, w, h et z qui correspondent à la position de la toile). Si cette condition est vérifiée nous ajoutons une instance *strokes* à la liste de *strokes* (*canvas*) avec les coordonnées et la couleur du *brush* à l'instant. [2]

[2] *la fonction check_new_line() sera expliquée dans le III) Optimisations*

```
if (on(x, x + w, y, y + h, - 1000, z) && pbrush.x != 0 && check_new_line()) {

  canvas.add(new strokes(pbrush.x, pbrush.y, brush.x, brush.y, 6, paint));
}
```

Enfin pour afficher le dessin nous avons utilisé une boucle “for” améliorée

```
192     for (strokes s : canvas) {
193       s.display();
194     }
```

L’avantage d’une ArrayList au lieu d’un simple Array est qu’une ArrayList est une liste d’objets flexible, c’est à dire qui n’a pas de taille initiale au contraire des Arrays.

III) Optimisations et difficultés rencontrées lors du projet

A) Améliorations apportées :

L’aspect de ce projet dont nous sommes le plus fiers ce sont les améliorations et optimisations au programme que nous avons pu mettre en oeuvre au fur et à mesure de notre avancement.. Ayant réalisés avant l’ISN quelques projets finis, ce projet était le premier où on s’est posée la question, une fois que le programme ou algorithme marchait, de comment l’améliorer.

1. zone / amélioration de l’algorithme de détection

Afin d’accélérer la vitesse de l’algorithme de détection, au lieu de chercher parmis tous les pixels des images de la caméra, (1280*720), il est plus efficace de se concentrer sur une zone de recherche. Celle-ci est déterminée par rapport à la position de l’objet recherché sur l’image antérieure. Puisque nous avons 4 objets à détecter (tête et pinceau de chaque caméra), nous avons créé un tableau d’entiers appelé zone[]. Sa taille initiale est de 16 entiers, car nous avons besoin de 4 paramètres pour définir la zone de recherche de chacun de ces objets.
Une fois les valeurs de zone[] initialisées dans le setup() (par groupe de 4 valeurs étant initialisées à 0, 0, 1280, 720) *[paint_pc l.54]*, nous avons besoin d’identifier chaque objet avec une variable “trackID” qui sera un paramètre de la fonction “*detect*” et prend les valeurs 0, 1, 2, 3 correspondant aux objets *[paint_pc l.143]*. Ce qui explique pourquoi on appelle chaque zone dans la fonction “*detect*” avec “+ trackID*4” dans son indice. EXPL : zone[1 + trackID*4].
Enfin nous définissons une dernière variable modifiable en temps réel, appelée zone_length, qui nous aidera à définir la zone de recherche (un carré autour de l’objet de côté 2*zone_length) une fois que la position de l’objets sera trouvée.

```
if (n_pixels > 50) {

  vector.x = cam.width - (total_x/ n_pixels);
  vector.y = total_y/ n_pixels;

  zone[0 + trackID*4] = int((total_x/n_pixels)- zone_length); //definit la zone de recherche pour la prochaine execution de la fonction
  zone[1 + trackID*4] = int(vector.y - zone_length);

  zone[2 + trackID*4] = int((total_x/n_pixels) + zone_length);
  zone[3 + trackID*4] = int(vector.y + zone_length);

  if ( zone[0 + trackID*4] < 0 ) zone[0 + trackID*4] = 0; //verifie que la zone ne sort pas hors des limites de l'image.
  if ( zone[2 + trackID*4] > cam.width) zone[2 + trackID*4] = cam.width;

  if ( zone[1 + trackID*4] < 0 ) zone[1 + trackID*4] = 0;
  if ( zone[3 + trackID*4] > cam.height) zone[3 + trackID*4] = cam.height;
} else {

  zone[0 + trackID*4] = 0; // si rien n'est trouvé dans la zone réduite, alors la zone redevient la taille de l'image.
  zone[1 + trackID*4] = 0;

  zone[2 + trackID*4] = cam.width;
  zone[3 + trackID*4] = cam.height;
}
```

2. la fonction check_new_line()

Puisque la boucle draw() est exécutée 60 fois par seconde, avec la méthode de dessin existante (voir II) B) 2)), nous pourrions facilement ajouter à la liste 60 lignes identiques par seconde sans que le joueur ne bouge le pinceau / brush.
Pour éviter que le tableau de lignes (*canvas*) ne devienne extrêmement long, alors que le joueur ne dessine pas de nouvelle ligne, il nous fallait une condition qui vérifie si la ligne dessinée était nouvelle. C'est le rôle de la fonction check_new_line().
Cette fonction renvoie un booléen et va comparer la position x et y des lignes précédemment enregistrées dans la liste "canvas" avec la position x et y actuelle du pinceau/brush. Si elles sont égales alors la fonction renvoie "false" et n'ajoute pas de ligne à la liste canvas, et donc ne dessine pas de nouveau trait.

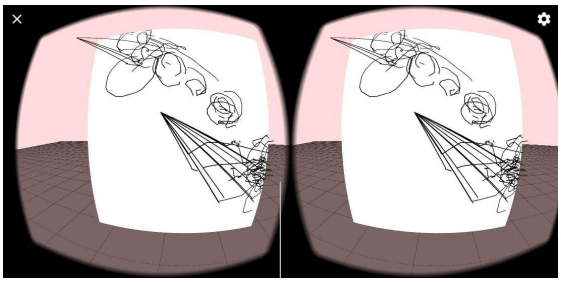
```
boolean check_new_line() {

  boolean permission = true;

  if (canvas.size() > 10) {

    strokes s = canvas.get(canvas.size() - 1);

    if (s.x == brush.x && s.y == brush.y) {

      permission = false;
    }
  }

  return permission;
}
```

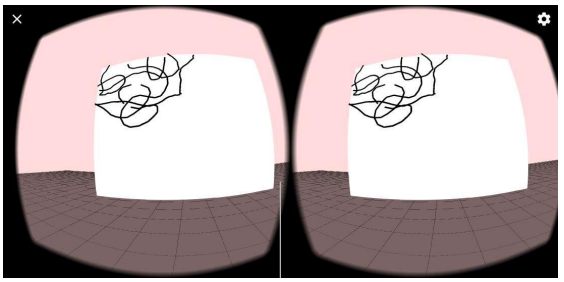
B) Difficultés rencontrées :

Nous avons rencontré beaucoup de difficultés dans l'exécution de ce projet. Le principal défi du projet a été d'acquérir les connaissances sur des aspects essentiels pour sa réalisation: la 3D, les calculs de distance en stéréoscopie et le networking. Nous avons passer la plupart de notre temps sur des pages de référence (cf. sources) pour apprendre sur ces nouveaux domaines.

1. Difficultés au niveau du programme.

Nous avons expérimenté des soucis de production et d'efficacité tout au long du développement, comme par exemple la taille maximale des messages envoyés dans le networking, l'environnement du téléphone, la calibration des coordonnées et la zone.
Lors de notre premier assemblage du projet, le 16 janvier, (première fois qu'on avait mis ensemble un algorithme de détection + dessin rudimentaire), on a compris la nécessité d'avoir une fonction qui calibrait les valeurs avant de les envoyer sur le téléphone. Voici nos premiers dessins :





2. Difficultés au niveau du matériel et du travail

Un des problèmes avec le matériel que l'on avait, était le fait qu'on a utilisé le même modèle de caméras (Logitech c270), pour avoir les mêmes angles de focalisation. Néanmoins, l'ordinateur identifiait les deux caméras comme étant une seule même caméra. Nous avons été amenés à effectuer quelques changements dans l'éditeur de registre de Windows (regedit) pour résoudre ce problème.
Une autre difficulté était que nous étions limités à 2 webcams, et comme seulement l'un d'entre nous avait les caméras par lapse de temps, l'autre était empêché de travailler en parallèle.
Enfin, la mise en commun des programmes, après avoir eu des développement séparés, est apparu comme une des problématiques majeures. Avec l'expérience acquise au cours de ce projet, nous n'hésiterions pas à mettre en place des outils de mise en relation automatique comme "git", qui nous a été recommandé. Cela nous aurait évité de passer beaucoup de temps à essayer de fusionner différentes versions des programmes.

3. Améliorations ultérieures possibles

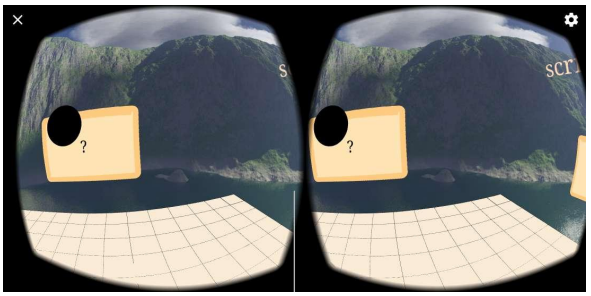
La version du projet tel que nous le présentons aujourd'hui répond parfaitement à nos objectifs initiaux. Mais, comme beaucoup de programmes, de multiples extensions sont possibles, notamment celles identifiées en fin du développement. Les extensions suivantes pourraient compléter le projet :

- ajouter un système de mélange des couleurs avec les seaux afin d'obtenir toutes les couleurs du spectre RVB.
- ajouter du son, pour l'ambiance, mais aussi lors d'interactions avec, par exemple, les seaux de peinture.

IV) Pour aller plus loin et impact sociétal

A) La création d'un cadre

A la fin du développement, nous nous sommes rendu compte que avec les caméras et le programme que nous avons mis en place, nous avions créé un cadre, une structure, pour développer des jeux en réalité virtuelle. Ce cadre nous a permis de développer la simulation de peinture en réalité virtuelle, une des applications les plus directes de contrôle d'un objet dans l'espace. Nous nous sommes aussi rendus compte que nous pouvions utiliser la plateforme que nous avions créée pour d'autres types d'application.
Pour illustrer à quel point la création de jeu à partir du système mis en place est facilité, nous avons inséré le jeu du chat (un jeu que nous avons appris à faire en début d'année en cours d'ISN) dans le programme du téléphone (paint_vr). Le but du jeu est de retourner des cases afin de trouver le mot "CHAT". Nous avons implémenté le jeu du chat dans ce programme : il est accessible dans le menu via un bouton caché à gauche du joueur.



B) Impact sociétal

Il est évident que notre projet n'a pas d'impact direct sur la société , mais il témoigne de l'avancé de la VR dans le monde de l'informatique. La réalité virtuelle est un secteur de l'informatique qui se développe rapidement avec ses applications dans différents domaines comme la thérapie, l'architecture, les films, la formation, et bien sûr les jeux-vidéos. Dans l'esprit collectif, le réalité virtuelle est souvent une expérience demandant beaucoup de ressources: des installations spécifiques, un ordinateur puissant et des périphériques coûteux. Notre projet montre qu'il peut exister des expériences de réalité virtuelle avec relativement peu de moyens, et que le futur de ce média, notamment grâce au développement rapide des smartphones, pourrait très vite se trouver dans nos poches.

Sources et Liens importants :
les calculs de coordonnées :
http://www.warse.org/pdfs/2013/icctesp02.pdf
le réseau :
http://ketai.org/tutorials/
http://www.sojamo.de/libraries/oscP5/#resources
réalité virtuelle :
https://processing.org/reference/
https://android.processing.org/reference/index.html

Sources d'inspiration :
https://www.youtube.com/user/shiffman
skyboxes:
https://gamebanana.com
Toutes les skyboxes sont libres de droit, le logo scribbles in VR et les modèles 3D sont fait maison, le jeu du chat est la propriété intellectuelle de M. Grava.

Remerciements: Nous tenons à remercier M. Camponovo et M. Grava, ainsi que nos parents qui ont corrigé les nombreuses fautes d'orthographe de ce rapport.